DISEÑO Y CONTRUCCIÓN DE UNA MAQUETA ROUV PARA LA INSPECCIÓN SUBACUÁTICA "*NAUTIL*"

Trabajo Final de Grado

Facultad de Náutica de Barcelona
Universidad Politécnica de Catalunya

Trabajo realizado per:
VLADYSLAV LENKO

Dirigido per:
ROSA MARIA FERNANDEZ CANTÍ

Doble titulación de Grado en Ingeniería en Sistemas y Tecnología Naval y Grado en Tecnologías Marinas

Barcelona, 29 de octubre de 2020

Departamento de Ingeniería de Sistemas, Automática e Informática Industrial

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Nàutica de Barcelona

Agradecimientos

Agradezco la colaboración de todos los que han hecho de este trabajo un trabajo mejor. En especial a Rosa, Víctor, Josep, Marc, Joan, a mi madre y a Fer, Alex, Pol, Ignasi y a todos aquellos que no he nombrado en esta lista pero han colaborado de una u otra manera en hacer de este trabajo algo mejor.

También un agradecimiento a todos los que comparten sus conocimientos en la red y publican tan buenos blogs, videos y comentarios para que los demás podamos aprender de ellos con solo dar un click.

Sin todos ellos este trabajo no hubiera podido llegar a ser lo que es, un prototipo de sumergible, que pese a las muchas mejoras y aprendizaje por delante es funcional.

Resumen

En este proyecto se aborda el diseño y la construcción de un sumergible multifuncional, que entre otros permita realizar una inspección visual, recogida de muestras, y toma de variables para monitorizarlas. Todo esto utilizando un control y recepción de datos intuitivos y fáciles de usar que permitan al usuario operarlo sin gran dificultad.

Se empieza con un capítulo de introducción donde se describe la historia de los sumergibles desde sus inicios hasta el día de hoy, la motivación por realizar el proyecto, objetivos, planificación y organización.

Luego se sigue con las especificaciones y diseño del casco, se presenta el estado del arte y se realiza una valoración de los prototipos presentados. A partir de ella, se definen las especificaciones que debe satisfacer el ROUV para poder cumplir los objetivos del proyecto.

En la parte de operación se habla en primer lugar de las alternativas de operación del *ROUV.* A continuación, se explica lo que se refiere a la configuración que siguen los componentes del *hardware y software.*

Se seleccionan los accesorios y se explican ciertas características e implementación de los accesorios que son necesarias para su correcto funcionamiento.

En lo que se refiere al montaje se describe paso por paso como fue montado el *ROUV* en su totalidad.

Finalmente se describen las pruebas realizadas, la estática y la dinámica. Tras ello se expone el coste del proyecto, las conclusiones y los anexos correspondientes.

Abstract

This project deals with the design and construction of a multifunctional submersible, which allows a visual inspection, sample collection, and taking of variables to display them. All this using an intuitive and easy-to-use control and data reception that allows the user to operate it easily.

It begins with an introductory chapter where the history of submersibles is described from its start to the present day, the motivation for carrying out the project, objectives, planning and organization.

Then it continues with the specifications and design of the body, the state of the art is presented and an evaluation of the prototypes is carried out. Based on it, the specifications that the *ROUV* must have in order to meet the project objectives are defined.

In the operation part, the ROUV operation alternatives are discussed first. The following explains how to configure the hardware and software components.

The accessories are selected and certain characteristics and implementation of the accessories that are necessary for their correct operation are explained.

As regards assembly, a step-by-step description of how the ROUV was constructed in its entirety is described.

Finally, the static and the dynamic tests are described. After that, the cost of the project, the conclusions and the corresponding annexes are exposed.

Contenido

Capítulo 1. Introducción

1.1.Motivación

Los *ROUV's* (acrónimo del inglés *Remote, Operated, Underwater Vehicles)* generan/constituyen una industria en auge debido a la necesidad de realizar cada vez más inspecciones submarinas (cada vez las normativas son más estrictas debido al exponencial aumento de la información y tecnología). Además de que se produce un significativo ahorro en coste respecto a las inspecciones convencionales en vehículos tripulados o submarinistas

Según (B.Robert, 2020), se calcula que puede reducir hasta en un 80% los costes de las intervenciones submarinas de las plataformas 'off-shore' –en alta mar– de extracción de petróleo y gas.

La industria avanza y se demanda un aumento de las prestaciones y disminución de los costes finales de estos vehículos.

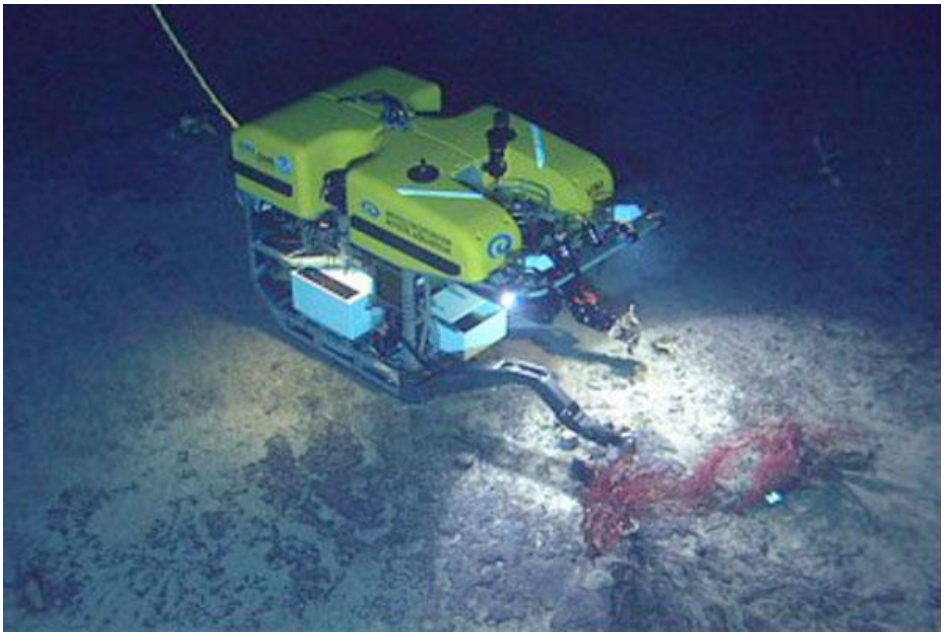


Figura 1. *ROUV Hercules* recogiendo muestras del lecho marino (OER, 2020)

1.2. Antecedentes y definición de los sumergibles

Antes de la aparición de los actuales ROUV'S se utilizaron los submarinos tripulados.

Uno de los vehículos submarinos más antiguos fue el diseñado por Cornelius Van Drebble cerca del 1620. Cien años después, durante la guerra civil de Estados Unidos el *Turtle*, de David Brushnell realizó el primer ataque submarino de nuestra historia. Más tarde fue desarrollado el *Nautilus* de Robert Fulton, el cual era propulsado por velas en la superficie y movido mediante unos pedales debajo del agua. (Allmediguer, 2020)

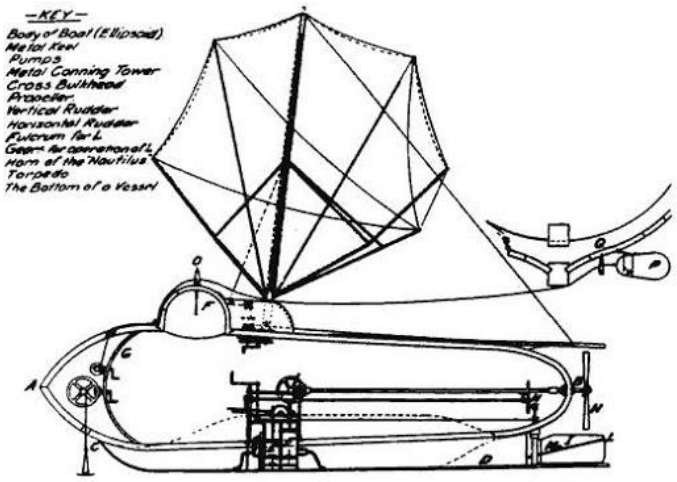


Figura 2. Nautilus de Robert Fulton (NavSource, 2020)

A finales del siglo XIX empezaron a aparecer los que se consideran los submarinos modernos. El *USS Holland* ya empezó a utilizar un motor de combustión interna para su propulsión. Durante la primera guerra mundial, los submarinos alemanes causaron importantes estragos a la flota británica y se apreció que el submarino era un arma de guerra de vital importancia.

El siguiente paso fue intentar reducir la resistencia a la formación de olas mejorando la forma del casco y timón. En la década de los años 30 se desarrolló el sumergible *Bathysphere y Benthoscope*, estos contactaban con la superficie mediante umbilical y tenía como función bajar a grandes profundidades. De la misma idea apareció el *FNRS-2* el cual tenía como lastre unos grandes tanques llenos de gasolina, también disponía de un sistema electromagnético para controlar la flotabilidad en grandes profundidades, el las cuales la presión es muy importante y la gasolina se comprime. Estos últimos sumergibles bajaban a profundidades mayores de 4500 metros.

Figura 3. Sumergible *Bathysphere* (Phatowary, 2020)

Es aquí cuando se finaliza el desarrollo del primer ROUV*,* el *Poodle*, construido por Dimitri Rebikoff en 1953 con fines arqueológicos. Este vehículo era controlado mediante una consola en la superficie y un umbilical.

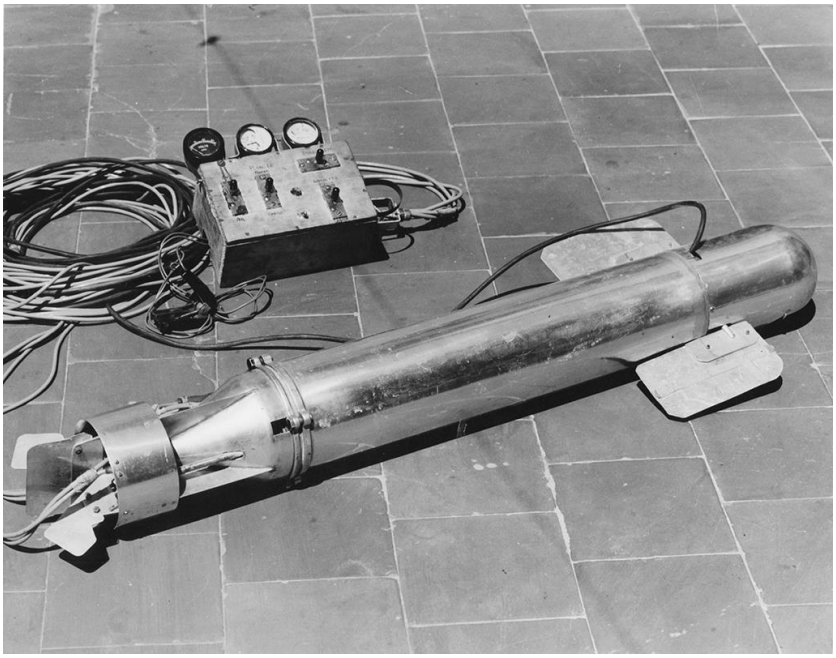


Figura 4. *ROUV Poodle* (Rebikoff-Niggler foundation, 2020)

En 1960, el sumergible *Trieste* de Piccard realizó un descenso récord de 10915 metros de profundidad en la fosa de las marianas.

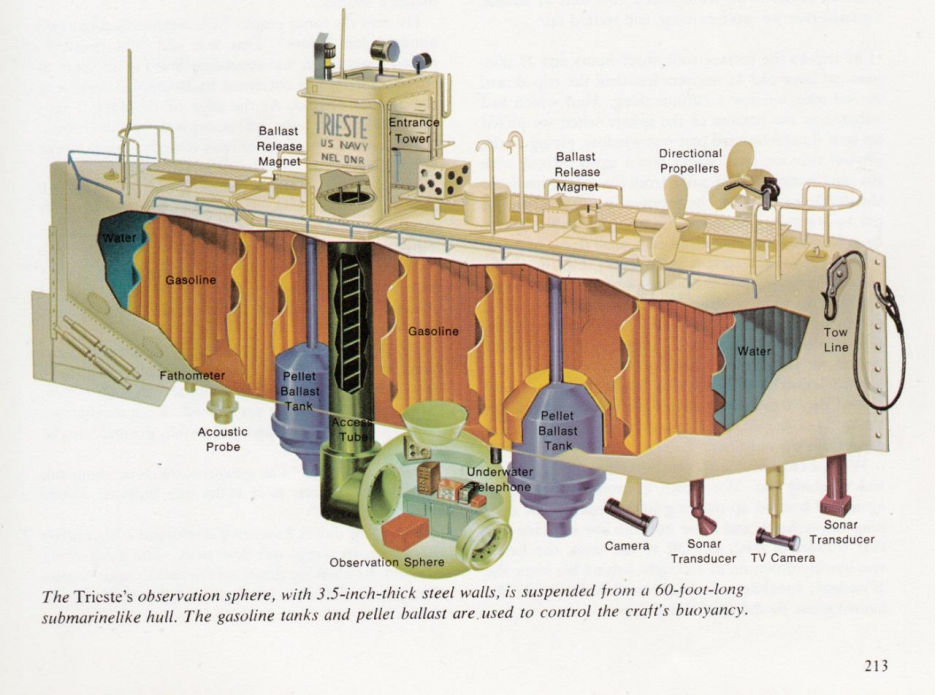


Figura 5. Corte del *Trieste* con sus partes (Anxel, 2020)

Más tarde fue diseñado el primer sumergible de aguas poco profundas, el *Denise,* de Jacques-Yves Cousteau, utilizado para exploraciones biológicas.

En 1963, con el hundimiento del submarino de propulsión nuclear *Thresher* se hizo grandes esfuerzos para conseguir desarrollar sumergibles de rescate. De este acontecimiento surgieron los denominados vehículos sumergidos de rescate *DSRV-1 y DSRV-2* operados por el ejército de los estados unidos.

Figura 6. *DSRV-1* instalado abordo de un submarino (Arbeam, 2020)

New York Times.

LATE CITY EDITION

NEW YORK, THURSDAY, APRIL 11, 1963. TEN CENTS

ATOM SUBMARINE WITH 129 LOST IN DEPTHS 220 MILES OFF BOSTON; OIL SLICK SEEN NEAR SITE OF DIVE

Thresher, nuclear-powered attack submarine, commissioned Aug. 3, 1961, at Portsmouth (N. H.) Naval Shipyard

THRESHER HUNTED

Rescue Craft Search Area of Last Test in 8,400-Foot Water

House Votes Works Plan; Backs President, 228-184

Figura 7. Titular sobre el hundimiento *USS Thresher* (Weckhunter, 2020)

En el mismo año, la marina de los EEUU con el fin de recuperar armamento del fondo marino desarrolló el *ROUV CURV.* Este vehículo demostró una utilidad sin precedentes y muchas naciones empezaron a desarrollar sus propios *ROUV* en los años venideros. Vehículos como el *Snoppy y Kaiko* siguieron esta línea. Durante los próximos años se siguieron botando sumergibles como la clase *Star, Alvin y Sea Cliff.*

En la siguiente etapa los sumergibles pasaron a ser controlados desde el buque de apoyo o base. Es decir, el piloto se encontraba en ambiente de una atmosfera mientras los "submarinistas" se encontraban dentro del sumergible. Esto permitía un ahorro considerable de la energía de los "submarinistas" con un consiguiente mayor tiempo de inmersión. Los sumergibles *Perry PC 1801 y Jhonson Sea Link* seguían esta metodología.

Con el paso de los años ha crecido la demanda de realizar inspecciones y reparaciones en zonas de mucha profundidad como es el caso en la industria petrolífera, por lo que se ha apostado por vehículos autónomos conectados mediante umbilical. A día de hoy hay una variedad muy grande de sumergibles, muchos de los cuales son *ROUV*.

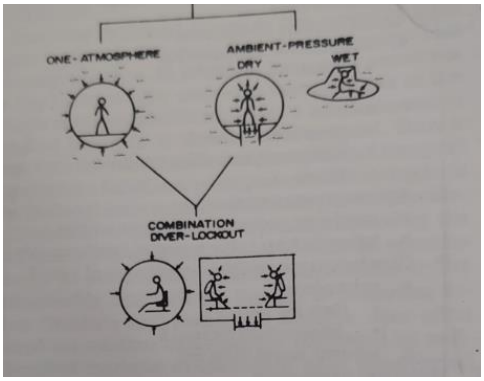


Figura 8. Forma de controlar los sumergibles (Allmediguer, 2020)

Uno de los últimos hitos en lo que se refiere al uso de los vehículos *ROUV* es la llegada de los hombres al fondo de la Fosa de las Marianas, en 2012 *James Cameron* a bordo del *Deepsea Challenger* y en 2019 *Vescovo* y *Haley* consiguieron bajar a una profundidad récord de 10928 metros en la misma fosa.

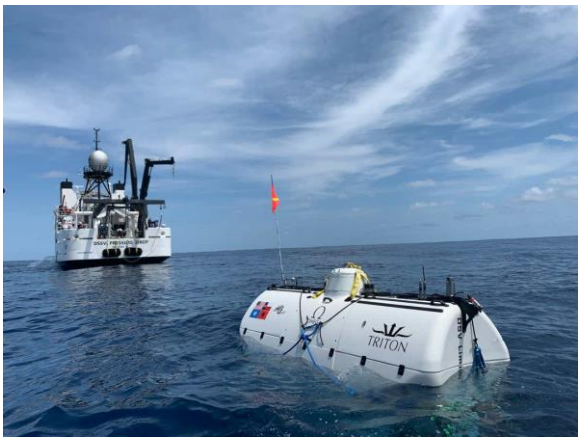


Figura 9. Expedición de la Fosa de las Marianas (Killer, 2020)

Figura 10. *KAIKO ROUV* (&JunH, 2020)

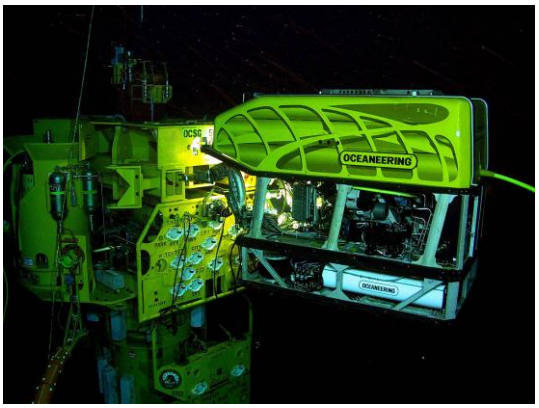


Figura 11. *ROUV* de *Oceaneering* (oceaneering, 2020)

Como ya hemos visto, podemos definir a los *ROUV* por el significado de sus iniciales, que es el de vehículos subacuáticos operados remotamente. Ahora bien, hay varias formas de clasificarlos (funciones que realizan, sector al que se ofertan, profundidades a las que bajan…) pero aquí los clasificaremos según la forma de operarlos:

1. Conectados al exterior mediante umbilical: En la actualidad conforman la mayoría de vehículos. Esto es debido a la buena capacidad de transmisión de datos y la posibilidad de que estos vehículos sean de un tamaño y complejidad más reducida que los inalámbricos.

   Dentro de esta categoría se sub clasifican en:

   a. *Towed (*arrastrados y sin propulsión propia).
   b. Clásicos (usando umbilical y con propulsión propia)

2. Conectados al exterior de forma inalámbrica: Conforman una minoría de *ROUV,* eso es debido a su complejidad (receptores abordo, transmisores, alimentación…) y al hecho de que es muy difícil transmitir señales a través del medio acuático. Como ventaja tienen que su libertad de movimiento es completa y no existe el obstáculo del umbilical.
   a. Pre-programados
   b. Controlados acústicamente

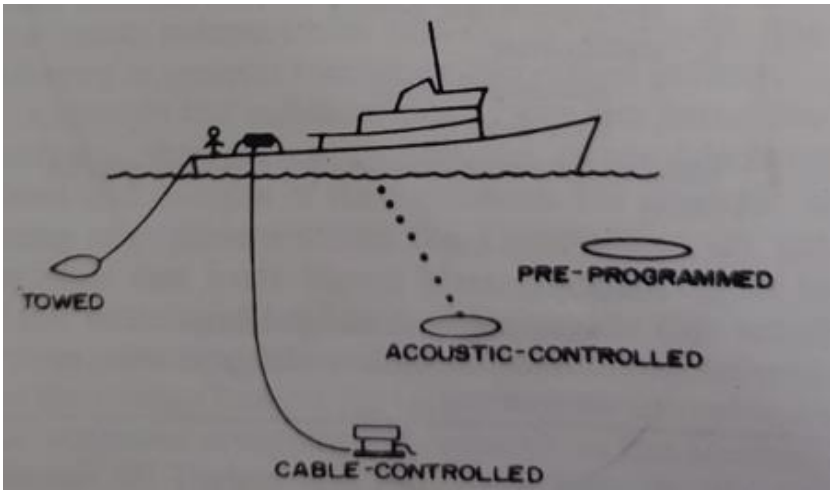


Figura 12. Clasificación de los *ROUV's* (Allmediguer, 2020)

1.3. Objetivos del trabajo

El objetivo general de este proyecto es diseñar y construir un ROUV, que hemos bautizado como NAUTIL, basado en algunos prototipos ya existentes que pueden verse en 2.1.2 Prototipos existentes, a los cuales aplicaremos mejoras que le permitan desenvolverse mejor en el medio acuático y realizar ciertas

funciones de observación y transmisión de parámetros. La planificación de su construcción y su construcción será lo más eficiente posible para los recursos y limitaciones que disponemos.

Todo el proceso será descrito en la memoria del trabajo en forma de manual para que luego, la persona que lo desee pueda partir del diseño del *NAUTIL* y reproducirlo o mejorarlo.

Objetivos específicos:

1. Documentar proyectos similares que hay tanto en la red como en otros tipos de fuente para ver la viabilidad del proyecto tanto en tiempo, conocimientos y presupuesto.

2. Definición de las especificaciones que queremos que tenga nuestro *ROUV* para poder planificar tareas, fechas y presupuesto aproximado.
3. Diseño y construcción del casco.
4. Selección de la instrumentación e implementación electrónica.
5. Elaboración de un presupuesto final.
6. Realización de pruebas experimentales
7. Extracción de conclusiones.

1.4. Planificación del trabajo

Antes de empezar el proyecto hemos prestado especial atención a la fase de búsqueda de información. La búsqueda ha cubierto los aspectos de estudio de proyectos publicados en la red, el análisis de sus resultados y nivel de dificultad.

Una vez hecha la búsqueda previa hemos seleccionado un conjunto de modelos que podríamos realizar y finalmente hemos seleccionado un finalista para seguir como base.

A continuación, se muestra un diagrama *GANTT* para organizar las diferentes fases del proyecto. Cabe destacar que al ser un proyecto pensado para hacerlo una sola persona los pasos de trabajo son lineales, es decir, no habrá ningún momento en el cual simultáneamente se estén haciendo dos pasos (a excepción de la compra ya que esta se espera que demore algo). Esto permite un mejor control del proceso de fabricación, pero como contra tiene que el plazo de entrega del proyecto aumentará considerablemente.

1 MEC. tarea_0 COMPRA ESTRUCTURA, MOTORES	17/02/20	5/03/20
1 MEC.tarea_1MONTAJE ESTRUCTURA PVC, MOTORES	6/03/20	19/03/20
1 MEC.tarea_2Compra accesorios(CAMERA Y DIODOS SI HACE FALTA ,SENSORES,BATERIA,COM...	17/02/20	17/07/20
2 ELECTRICAL tarea_3Compra command box+interrupotores,cables+rs232 y payload box(compo...	17/02/20	27/03/20
2 ELE.tarea_4Montaje command box	30/03/20	8/04/20
2 ELEC.tarea_5Montaje payloadbox y soldadura de sus componentes	9/04/20	6/05/20
3 TRON.tarea_7Programar con Arduino	21/05/20	17/06/20
4 COMM.tarea_8Probar comunicación serial	18/06/20	10/07/20
4 COM.tarea_10Integrar Arduino con el Hardware	8/07/20	28/07/20
5 Signal processing.tarea_11Matlab y instalacion de la camara	29/07/20	25/08/20
5 Signal processingtarea_12 Representacion gráfica por ordenador	26/08/20	8/09/20
6 FINISHtarea_13 Ensamblaje	9/09/20	29/09/20
6 FINISH tarea_14 Tests	30/09/20	20/10/20
1' MEMORIA	17/02/20	29/10/20

Tabla 1. Diagrama de *GANTT*

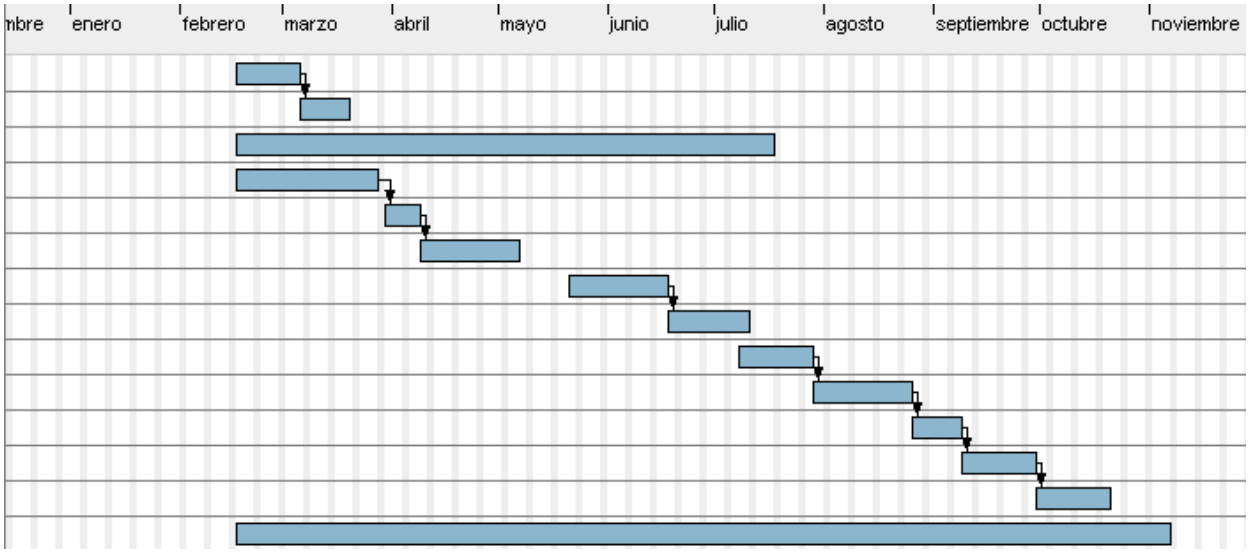


Figura 13. Diagrama de *GANTT*

Debido al confinamiento causado por la pandemia del *coronavirus,* los plazos de entrega del suministro de materiales se han visto afectados. Aun así, el sistema de envíos por internet funcionó y se pudo obtener todo el material necesario para el proyecto.

1.5. Organización de la memoria

Hemos querido que la memoria de este proyecto siga unas pautas concretas y que permita ser usado como manual para la construcción de un *ROUV.* Por lo tanto, gran parte de la memoria consistirá en los diferentes pasos para poder construirlo.

Así pues, esta memoria está estructurada de la siguiente forma:

En el Capítulo 1. *Introducción*, se expone la motivación del trabajo, se presenta una panorámica del campo de los sumergibles en general y tipos de *ROUV* en particular, y se listan los objetivos que se pretenden conseguir al tener el trabajo realizado. También se planifican los plazos y diferentes puntos de los cuales está compuesto el proyecto final para no perderse en el proceso de la fabricación.

En el Capítulo 2. *Especificaciones del NAUTIL y diseño del casco*, se presenta el estado del arte y se realiza una valoración de los prototipos presentados. A partir de ella, se definen las especificaciones que debe satisfacer el ROUV para poder cumplir los objetivos del proyecto. Además, se nombran una serie de limitaciones de las que disponemos para realizar nuestro proyecto y las cuales nos permitirán acotar aún más el proyecto y hacerlo más concreto. Para finalizar, se describe el proceso seguido para diseñar el casco.

Capítulo 3. Operación. En este capítulo se habla en primer lugar de las alternativas de operación del *ROUV* y se escoge la más conveniente. A continuación, se explica lo que se refiere a la configuración que siguen los componentes del microcontrolador *ARDUINO*. Luego se habla de la interfaz que se configura entre el microcontrolador y un ordenador para poder monitorizar ciertos parámetros que mide el *ROUV y* la imagen que proporciona la cámara. Esto se consigue con el programa *MATLAB*.

Capítulo 4. *Accesorios e instrumentación*. En este capítulo se seleccionan los accesorios necesarios para cumplir el objetivo de nuestro *ROUV.* A continuación, se explican ciertas características e implementación de los accesorios que son necesarias para su correcto funcionamiento.

Capítulo 5. Montaje, donde se explica paso por paso como fue montado el *ROUV* desde el montaje del cuerpo exterior hasta el montaje y sellado de la *payload box.* Todo esto pasando por el añadido de los motores, accesorios y montaje de la *command box.*

Capítulo 6. Pruebas experimentales: En primera instancia se realiza la prueba estática en la cual se quitan todos los elementos que puedan estropearse debido a una entrada de agua en la *payload box* y se sella esta como si el proyecto estuviera acabado. Luego se sumerge el *ROUV* manualmente y se comprueba que la estabilidad sea la correcta. Al sacar el artefacto se comprueba que no hay ninguna entrada de agua. La prueba final es la dinámica, en la cual se reinsertan todos los componentes electrónicos y se sumerge el *ROUV,* esta vez ya a distancia. Se hace una inmersión en espiral para comprobar y/o modificar la flotabilidad.

Capítulo 7. Presupuesto, donde se presenta un presupuesto de los componentes utilizados.

Capítulo 8. Conclusiones y líneas futuras. Se listan las principales conclusiones del proyecto y se sugieren futuras líneas de trabajo.

Los anexos están clasificados de la siguiente forma:

1. Códigos completos de *ARDUINO y MATLAB* utilizados.
2. Instrucciones de usuario.
3. Ficha de análisis de fallos donde se exponen los fallos más probables que tenga el sumergible y su gravedad y solución. Este anexo, junto a las instrucciones serán de gran utilidad para el usuario final.

Capítulo 2. Especificaciones del NAUTIL y diseño del casco

En este capítulo y siguientes se abordarán los distintos aspectos del diseño. Antes de seleccionar cada parte se realiza una breve descripción del estado del arte a fin de justificar la elección.

En particular, en este capítulo se presentan y analizan diversos prototipos ya existentes y, a partir de dicho estudio, establecemos las especificaciones que debe satisfacer nuestro *ROUV* tanto de estructura y operación como de funcionalidad.

2.1 Estado del arte

La idea inicial es realizar un prototipo de pequeño tamaño, sumergible en agua dulce, con umbilical y dotado de sensores para el studio del fondo.

A partir de nuestra idea inicial se ha realizado una búsqueda de todo tipo de proyectos similares y durante el camino se han descartado o adoptado ideas para nuestro futuro *ROUV.* A continuación, se van a exponer las principales ideas que fueron saliendo durante esta fase y el porqué de adoptarlas o descartarlas.

2.1.1 Teoría de submarinos

Esta sección, fue extraída de Capítulo 5: *Hydromechanical Principles* de (Allmediguer, 2020).

- **Fuerzas y momentos que actúan en un sumergible**

La siguiente lista incluye las fuerzas que actúan sobre un submarino.

1. Peso
2. Desplazamiento
3. Resistencia al fluido externo
4. Reacciones producidas por el sistema de propulsión
5. Fuerzas de contacto (viento, olas y objetos)
6. Resistencia a los cambios debido a la inercia

Relacionado con estas, cabe destacar que un sumergible ''cerrado'' es decir que desde todo su perímetro exterior hasta el núcleo tenga superficie mojada en contacto con el agua experimentará mayores fuerzas y momento respecto a otro que tenga espacios vacíos (como es nuestro caso, el cual entre el casco o jaula y la *payload box* tiene espacios libres por los que puede circular el agua).

Volviendo con las fuerzas anteriores, todas ellas actúan sobre un sumergible y pueden clasificarse en estáticas o dinámicas.

a. Fuerzas estáticas

Se incluyen el peso del submarino, que tiene un valor constante, y el desplazamiento, siendo este:

$$\Delta = V \cdot \gamma \qquad \textbf{(1)}$$

Donde Δ es el $Despazamiento$, V el $Volumen\ de\ Desplazamiento$ y γ la $Densidad\ del\ agua\ alrededor\ del\ submarino$.

Según la relación que existe entre estas dos fuerzas, peso (W) y desplazamiento (Δ) podemos definir tres tipos distintos de submarinos:

- Flotabilidad positiva, cuando el desplazamiento es mayor que el peso.
- Flotabilidad neutra, cuando las dos fuerzas son iguales.
- Flotabilidad negativa, cuando el paso es mayor que el desplazamiento.

Las dos fuerzas están alineadas cuando el submarino está en reposo. Cuando otras fuerzas causan una inclinación del submarino, el peso y el desplazamiento crean un momento adrizaste que se opone a la inclinación; a este efecto se le llama estabilidad estática.

b. Fuerzas dinámicas

Estas fuerzas incluyen las situaciones de *lift* (L) la resistencia (D) y el empuje (T).

L se utiliza para controlar el submarino en caso de poseer planos de control. En caso de que se haga mediante propulsores, la fuerza encargada del control del vehículo es T. La estabilidad del submarino proporcionada a través de estas fuerzas se conoce como estabilidad dinámica, que está muy ligada a la estabilidad estática.

Hay que tener en cuenta en la estabilidad dinámica el efecto de los elementos externos al submarino, como pueden ser las condiciones climatológicas o la fuerza que ejerce un cable umbilical sobre el submarino.

- ### Estados de equilibrio

Cuando todas las fuerzas, tanto dinámicas como estáticas son iguales a cero, podemos decir que el submarino está en un equilibrio total. El desequilibrio dinámico crea una aceleración en alguna de las direcciones. Un desequilibrio estático crea una inclinación en alguna de las direcciones.

Estos conceptos están relacionados con los grados de libertad: cuando los grados de libertad se reducen, las ecuaciones del movimiento se van simplificando hasta la familiar ecuación mecánica de la fuerza y del momento.

La estabilidad metacéntrica, producida por el peso y el desplazamiento, resiste las inclinaciones producidas por las fuerzas dinámicas. En los submarinos de baja velocidad esta debe ser lo suficiente para poder realizar trabajos con seguridad y no tan grande como para necesitar excesiva potencia de propulsión para la realización de maniobras. La estabilidad de movimiento es aconsejable en el plano horizontal y necesario en el plano vertical debido a que se puede perder el control de la profundidad. En los ascensos y descensos rápidos la estabilidad metacéntrica y la velocidad deben combinarse suficientemente bien para garantizar que las fuerzas que se ejercen sobre los submarinos son suprimidas.

- ### Categorías hidromecánicas de los submarinos

Respecto al diseño, los sumergibles atendiendo al aspecto de categorías hidromecánicas se clasifican como:

- Submarinos de flotabilidad neutra. En estos, el peso y desplazamiento se anulan produciendo dicha flotabilidad neutra. Pueden clasificarse como:

  - Submarinos con reserva de desplazamiento: Estos vehículos son capaces de variar su desplazamiento. Cuando están en superficie pueden tener una zona fuera del agua, esta característica facilita las operaciones tanto de entrada o salida como las de izado y descenso al agua.

  - Submarinos sin reserva de desplazamiento: Estos submarinos no varían su desplazamiento y por tanto siempre tienen la misma flotabilidad.

- Submarinos sin flotabilidad neutra. Son equipos más complejos y versátiles. Se clasifican en:

  - Soportados estáticamente: En esta categoría se podrían incluir submarinos que deben ser retenidos de su descenso mediante cable.

  - Soportados dinámicamente.

    - Soportados por empuje: En este caso son los propulsores los que generan una fuerza que junto al peso hace descender o ascender al submarino. Uno de los ejemplos interesantes que se encuentra en este grupo son los submarinos que no producen nubes al navegar a ras de fondo. Para ello, la flotabilidad del sumergible debe ser positiva y la cara de succión de los propulsores verticales debe estar tomando agua del fondo hacia la superficie.
    - Soportados por Lift: Este tipo de submarinos necesitan cierta velocidad para poder descender o ascender debido a que la fuerza resultante se obtiene mediante alerones que según su inclinación generan un aumento de presión.

- **Sistemas de control**

Se deben incorporar sistemas de control para que el sumergible pueda cambiar de forma eficaz y segura de una condición de operación a otra. Los principales sistemas de control que podemos encontrar son:

- Piloto/autopiloto
- Sistema de transferencia de ordenes
- Actuador
- Comparador y comunicador de acción al piloto o autopiloto.

Una forma de clasificar los sistemas de control es, por si son estáticos o dinámicos. Los estáticos permitiendo libertad en 3 ejes y permitiendo el trimado, escora y ascenso y descenso. En cambio, los sistemas dinámicos permiten los 6 grados de libertad.

2.1.2 Prototipos existentes

Se han seleccionado 4 posibles diseños:

El primer diseño corresponde al proyecto de la Figura 14. Diseño con batería interna . En este caso cabe destacar que se consigue el control de la propulsión gracias a un microcontrolador *ARDUINO* que desde

la *command box* transmite las órdenes de movimiento a los *motor controllers* y estos a los motores del artefacto. Una cosa a destacar en este proyecto es que la alimentación para la propulsión y cámara se encuentra dentro del *ROUV,* esto ayuda disminuir la flotabilidad ya que en sí el diseño es poco compacto con lo cual se necesita de lastre adicional para su correcto funcionamiento.

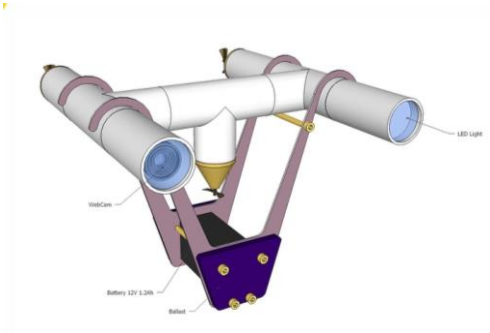


Figura 14. Diseño con batería interna (Geirandersen, 2020)

El diseño de prisma rectangular de la Figura 15. Diseño de Harry Bohm: (a) (b) es el más popular tanto en robots *amateurs* como en los comerciales. Dispone de dos flotadores en la parte superior y 2 cilindros en la parte inferior para su posado sobre suelo. Esto le da un aspecto de rigidez. Además, dispone de mucho espacio en la zona central para disponer la *payload box* y mucho espacio lateral para cámaras, motores y demás *gadgets.*

(a) **(b)**

Figura 15. Diseño de Harry Bohm: (a) (homebuilrtovs, 2020) (b) (Proto57, 2020)

En el caso mostrado en la Figura 16. Diseño por RC se ha realizado un proyecto de *ROUV* con un alto nivel de protección IP. El robot fue testeado en profundidades de hasta 10 metros sin problema alguno. Su peculiaridad principal es que los diseñadores han optado por un control mediante ondas, con lo cual se tiene el controlador RC que comunica con una antena y un radio receptor localizados en un flotador y estos ya si comunican mediante cable con el *ROUV.* La ventaja obvia de este sistema es que el robot puede desplazarse grandes distancias. El inconveniente es que resulta muy cara la comunicación RC cuando se desean monitorizar muchos parámetros como son la presión, cámara, propulsión y demás.

Figura 16. Diseño por RC (GRAVIT0N, 2020)

La Figura 17. Diseño compacto es un robot con 4 propulsores y controlado mediante un microcontrolador *ARDUINO*. *A* destacar de él que es un modelo compacto el cual tiene los tanques para modificar su flotabilidad en la parte inferior.

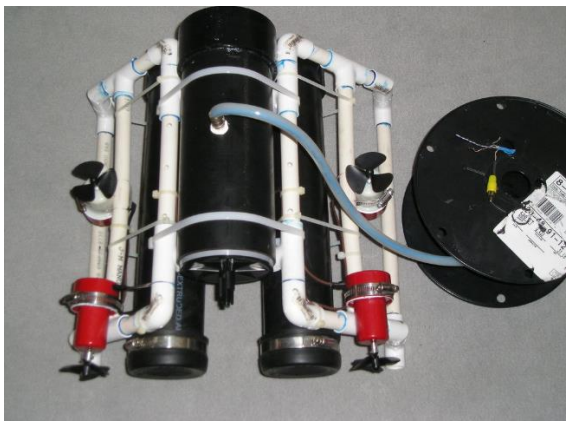


Figura 17. Diseño compacto (SpiceSHipOne, 2020)

2.1.3 Valoración de los diseños

Analizando los anteriores diseños empezamos descartando el diseño 3 debido a que la implementación de un control *RC* resultaría muy caro y complicado debido a todos los canales que se requeriría transmitir (cámara, sensores varios y demás extras).

Continuamos descartando el modelo 1 ya que tiene poco espacio para sensores, cámara y demás extras que podrían ponerse. Además, descartamos disponer una batería interna dentro del *ROUV.*

Entre los 2 diseños restantes nos quedamos con el segundo. Las razones son varias:

- La mayoría de *ROUV 's* que se han realizado siguen este diseño.
- Tiene mucho espacio disponible para tener juego en futuras mejoras.
- Está muy equilibrado.
- Gracias a sus dos tanques superiores no habrá problemas para conseguir una flotabilidad neutra.

2.2 Especificaciones del *NAUTIL*

Una vez analizados los diseños existentes y descartando algunas soluciones, las especificaciones del *NAUTIL* son las siguientes:

Funcionalidad:

- Se diseñará para ser sumergido en agua dulce. Ello evita los problemas de corrosión típicos de la operación en agua salada, aunque no supondría gran problema ya que el *ROUV* no dispondrá de piezas críticas que se puedan corroer al alcance del agua.

- Desplazamiento: Se deberá poder mover a una velocidad constante y el cambio de velocidad y sentido deberá ser lo más suave posible (por ello se implementarán *motor drivers* con el fin de regular la velocidad). Los grados de movimiento que deberá asumir el *ROUV* serán el avance/retroceso horizontal y el ascenso/descenso vertical. Como puede verse, el sumergible no podrá realizar el *roll* ya que este movimiento no aportaría nada más que la posibilidad de desequilibrar el aparato. En todo caso se instalará un acelerómetro y giroscopio que nos permitirá monitorizar tanto la velocidad de avance como la situación de equilibrio del aparato.

- Autonomía: Al ser un vehículo de inspección, se le dará una autonomía de 1 hora. Con lo cual adaptaremos la alimentación a las necesidades de la potencia máxima requerida para poder contar con esa hora. Cabe destacar que si precisamos de una mayor autonomía podemos simplemente conectar una fuente de alimentación o batería de mayor capacidad.

- Radio de acción y profundidad máxima: Debido a la pérdida de carga y limitaciones de longitud del USB (comunicación serial con *ARDUINO*) se decide, limitar la longitud del umbilical a 15 metros.
  Por otro lado, debido a que es un proyecto doméstico, la estanqueidad será equivalente al protocolo IP67 (garantiza que, dentro de los sitios críticos, no entrará agua en un periodo de 30 minutos a una profundidad de 1 metro).

Diseño (casco y estructura):

- Diseño compacto e hidrodinámico.

- Dimensiones: 65 de largo x 45 de alto x 40 de ancho (cm) serán las dimensiones máximas del *ROUV.*

- Estructura parachoques que evita impacto con equipo crítico del *ROUV*.

- Materiales plásticos en la medida de lo posible ya que son manejables y se portan bien con el agua salada.

Propulsión:

Se utilizan tres motores de corriente continua, 12 voltios y 3 amperios. En nuestro caso son motores extraídos de bombas de achique pequeñas, con lo cual ya están preparados para poder ser sumergibles. Al eje de estos se les añadirá una hélice para optimizar su capacidad propulsora. En la sección de referencias se incluye la hoja técnica de estos motores (vetus, 2020)

Figura 18 Bomba de achique J&N (Leroy M., 2020)

Alimentación

- Consumos

Para poder seleccionar la fuente de alimentación correcta se debe tener la suma de todos los consumos en las peores condiciones de trabajo y definir las características del cable umbilical que transportará la energía a bordo del vehículo. Iniciando este proceso de diseño, se incluye a continuación una tabla en la que se refleja el balance energético en la peor de las condiciones de operación.

Para averiguar el consumo de nuestra placa *ARDUINO*, vamos a realizar, con todos los componentes la medición del consumo con un Multímetro.

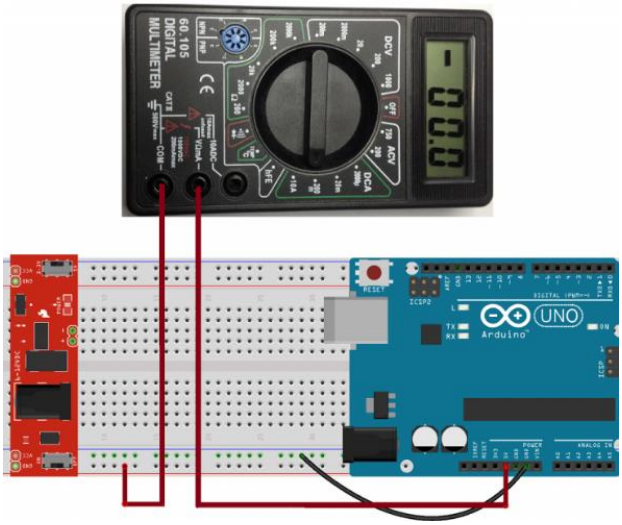


Figura 19 Cálculo de consumo de nuestro microcontrolador (Fuente: propia)

Componente	Voltaje(v)	Amperaje(A)	Unidades	Potencia(W)
Arduino Mega	12	0,2	1	2,4
Electroimán	12	1	1	12
Propulsores verticales	12	3	1	36
propulsores horizontales	12	3	2	72
Cámara y focos(*)	N/A	N/A	N/A	N/A
Total				122,4

Tabla 2 Balance eléctrico de nuestro ROUV

Cabe destacar que la cámara y los focos (*) no influirán en la selección de la alimentación ya que se ha decidido que tengan un circuito independiente debido a que se conseguirá una mayor calidad de imagen que con las cámaras compatibles con *ARDUINO.* Nuestra cámara dispondrá de una batería de 9 V y 5A que le permitirá tener una autonomía de 6 horas utilizando imagen y los focos.

- Caída de tensión del cable

Se selecciona una distancia operativa de 15 metros, con que la caída de tensión del cable viene dada por:

$$E = R \cdot I \quad \textbf{(2)}$$

Donde E es la caída de tensión, I es la intensidad de corriente que pasa por el cable, R es la resistencia del cable según su longitud y a su vez R es:

$$R = \rho \cdot \frac{\mathrm{L}}{\mathrm{S}} \quad \textbf{(3)}$$

Donde ρ es la resistividad del cable (en nuestro caso al ser un cable de cobre es de 0,0172 ($\Omega \cdot mm^2/m$) a 20º C, L es la longitud del cable y S es la sección del cable.

En nuestro caso:

$$R = \frac{0.0172\ \Omega\ \mathrm{mm2}}{m} \cdot \frac{15\ m}{1.5\ mm2} \quad \textbf{(4)}$$

Luego:

$$E = 0.172 \frac{\Omega}{m} \cdot 12A = 2.064\ V \quad \textbf{(5)}$$

Obtenemos que tendríamos una pérdida de 2 Voltios por el camino. Es un resultado satisfactorio ya que tanto los motores como electroimán funcionarán con esta pérdida.

En resumen, escogemos una manguera de 1.5 mm^2 de sección de cada cable de cobre a modo de umbilical.

Ahora solo falta escoger el número de cables que irán dentro de la manguera. Para nuestro proyecto necesitaríamos 8 (2 irían a la regleta que los distribuye hacia los controladores y 6 que serían para accionar manualmente los motores mediante los interruptores).

Instrumentación:

- Funciones añadidas: sensor de presión que revela la profundidad a la que se encuentra el ROUV, sensor de temperatura, compás, garra.
- Cámara
- Controlador *ARDUINO*
- Circuitos electrónicos: Acondicionadores de señal: amplificadores, filtros si se requieren.

Sistema de control de la propulsión y monitorización medidas:

- Desarrollo de una *Graphics User Interface* con *MATLAB (GUIDE-AppDesigner)*
- Comunicación *ARDUINO-MATLAB*

Económicas y temporales

- Que sea un ROUV funcional y de calidad, pero de un presupuesto reducido, es decir, no centrándose en aumentar el nivel de los aislamientos y comprando componentes de gama alta.
- Tiempo: inicialmente, disponemos de un plazo de unos 8 meses para realizar el proyecto

Todo el trabajo recae sobre 1 persona

Como ventajas podemos decir que la construcción de un *ROUV* es bastante popular y hay mucha información en la red acerca de diferentes proyectos. Con que no partimos desde cero.

2.3 Diseño y construcción del casco

Se ha escogido realizar el vehículo con una estructura prismática según lo mencionado en el apartado (2.1.3 Valoración de los diseños).

A continuación, se procede a diseñar a lápiz y luego en formato CAD como quedaría el producto final en lo que a formas y dimensiones se refiere.

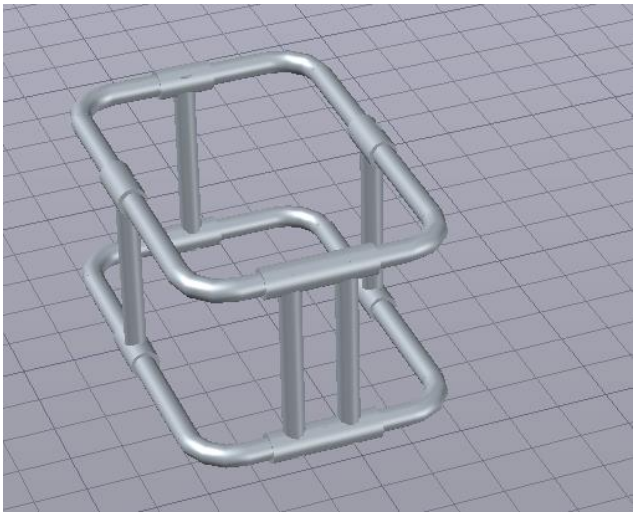


Figura 20 Diseño del cuerpo en CAD

Finalmente se decide reducir el tamaño del ROUV ya que en el primer diseño no se tuvo en cuenta la largura de los codos con lo cual, el sumergible quedó demasiado grande (75 de largo x 60 de ancho x 50 de alto (cm)).

En el segundo y definitivo modelo, además de incluir ya los tanques de lastre y tanques estancos para conseguir la flotabilidad neutra, se obtendrán las medidas definitivas del vehículo (60 de largo x 45 de ancho x 40 de alto (cm)).

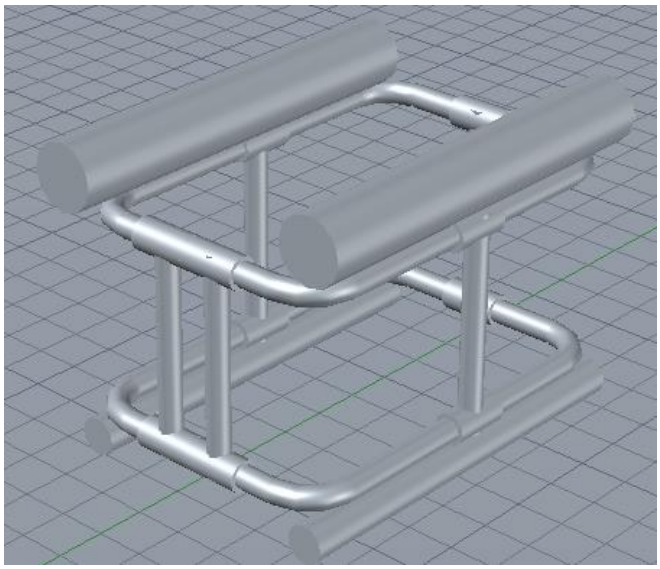


Figura 21 Versión final del sumergible en CAD

2.3.1 Resistencia al avance y potencia a suministrar

Como requisito pediremos una velocidad V de 1.5 kn para nuestro sumergible. Con lo cual, de fluidos hidrodinámicos sabemos que:

$$Rt = \frac{1}{2} \cdot Ct \cdot \rho \cdot S \cdot V^2 \quad \textbf{(5)}$$

donde ρ es la densidad del agua, es decir 1025 Kg/m^3 en nuestro caso, S la superficie mojada total, V la velocidad, es decir 1.5 kn, Ct es el coeficiente total, el cual esta compuesto únicamente por el coeficiente de fricción (Cf) debido a que nuestro sumerguible se desplazará completamente sumergido. Este se calcula así:

$$Cf = \frac{0{,}075}{(logRe-2)^2} \quad \textbf{(6)}$$

Para ello necesitamos el número de *Reynolds* (Re):

$$Re = V \cdot \frac{L}{\text{viscosidad}} = 0{,}77 \cdot \frac{0{,}45}{1.188 \cdot 10^{-6}} = 291666{,}6 \quad \textbf{(7)}$$

Ejemplo de cálculo: Si disponemos de un cilindro de L= 60 cm y D=32mm, la superficie es:

$$S = 2 \cdot Pi \cdot R \cdot L + 2 \cdot Pi \cdot R^2 = 0.061m^2 \quad \textbf{(8)}$$

Componente	**Superficie mojada**
Cilindro estructura de 32 mm de diámetro y 620 cm de longitud total	0,62 m^2
Cilindro estructura de 50 mm de diámetro y 120 cm de longitud total	0,19 m^2
Cilindro estructura de 90 mm de diámetro y 100 cm de longitud total	0,2 m^2
Caja estanca *Payloadbox* (175*175*100 mm)	0,1312 m^2
Motores y cámara	0,07 m^2
Total	1,30 m^2

A continuación, obtenemos

$$Rt = \frac{1}{2} \cdot 0.00624 \cdot 1025 \cdot 1.3 \cdot 0.77^2 = 2.46\text{N} \quad \textbf{(9)}$$

Teniendo este valor, podemos encontrar la potencia que se requiere para satisfacer la velocidad y resistencia anteriormente calculadas:

$$P = Rt \cdot V = 2{,}46 \cdot 0{,}77 = 1.9\ \text{N} \quad \textbf{(10)}$$

Como conclusión podemos sacar que, al pedir una velocidad de movimiento muy baja para nuestro *ROUV,* obtenemos una baja resistencia al avance y por consiguiente una pequeña potencia necesaria. Con que no tendremos problemas en este aspecto para seleccionar un motor.

2.3.2 Análisis de flotabilidad

Cálculo del desplazamiento y volumen desplazado

El principio de Arquímedes afirma que todo cuerpo sumergido en un fluido experimenta un empuje vertical y hacia arriba igual al peso de fluido desalojado para tender a su equilibrio. En otras palabras, el desplazamiento que produce el agua es igual al volumen desplazado multiplicado por la densidad.

$$\Delta = \nabla \cdot \rho \quad \textbf{(10)}$$

donde ∇ es el volumen desplazado por nuestro sumerguible en m^3 y Δ es el desplazamiento del agua en kg.

La Tabla 3 muestra el volumen total de nuestro ROUV.

A partir del volumen total, obtenemos que nuestro desplazamiento para agua dulce es

$$\Delta = 0.0184\ m^3 \cdot 997 \frac{kg}{m^3} = 18.34\ kg \quad \textbf{(11)}$$

Finalmente, a efectos prácticos y de estabilidad se ha optado por permitir el paso de agua de los cilindros 1 y 2 con que el desplazamiento ha quedado reducido a:

$$\Delta = 0.0110\ m^3 \cdot 997 \frac{kg}{m^3} = 10.98\ kg \quad \textbf{(12)}$$

Elemento	Longitud (mm)	Anchura (mm)	Altura (mm)	Diametro (mm)	Volumen (mm^3)
Cilindro (1)	6200			32	4986335
Cilindro (2)	1200			50	2356194
Cilindro (3)	1000			90	6361725
Caja estanca	175	175	100		3062500
Motores y cámara	120			50	1696460
Volumen Total (m^3)					0’0184

Tabla 3 Cálculo de volúmenes

Obtención de la flotabilidad neutra

En nuestro caso, el peso del *ROUV* sin lastre es de 4 kg con que deberemos lastrar con 6,98 kg para que la flotabilidad sea neutra.

Capítulo 3. Operación

En este capítulo se habla en primer lugar de las alternativas de operación del *ROUV* y se escoge la más conveniente. A continuación, se explica lo que se refiere a la configuración que siguen los componentes del microcontrolador *ARDUINO.* Luego se habla de la interfaz que se hace entre el microcontrolador y un ordenador para poder monitorizar ciertos parámetros que mide el *ROUV y* la imagen que proporciona la cámara. Esto se consigue con el programa *MATLAB*

3.1 Alternativas de operación

Hay 3 variantes en lo que se refiere a operar el vehículo:

- Usando relés que pongan en marcha/detengan los diferentes motores. Luego mediante un mando de RC modificado se controla por cable la propulsión del vehículo. La principal desventaja de este método es que nos limita bastante en el tema de que no podemos controlar y monitorizar los sensores de presión, electroimanes, etc. Su ventaja es clara, es el método más simple de operativa.

- Usando un microcontrolador (estilo *ARDUINO UNO)* podemos operar la propulsión gracias a unos *motor drivers*. Luego podemos además añadir los extras que nos permita nuestro microcontrolador (sensores de presión, temperatura, electroimanes, cámaras y demás). En este caso el vehículo también se controlaría desde una *control box* pero además deberíamos juntarlo con un lenguaje para poder, mediante una pantalla LCD monitorizar los parámetros de presión, luego profundidad, grabación, y demás extras. Esta opción nos permite también poder controlar los motores con un teclado y su pantalla. Claramente es más complejo hacer que todo funcione con este método, pero permite explotar mejor nuestro aparato.

- Usando un micro procesador (estilo *rasperry pi)* para el final control de la propulsión y demás parámetros. Como ventaja podemos decir que es el método que permite una mayor compatibilidad de equipos. Como inconvenientes decir que los micro procesadores son más caros que los microcontroladores.

Finalmente, tras una búsqueda exhaustiva hemos decidido optar por la segunda opción (usando un microcontrolador estilo *ARDUINO UNO*). Se ha descartado el uso de un micro procesador debido a que no hemos encontrado suficientes ejemplos realizados con este equipo como para sentirnos seguros y optar por este método.

Cabe destacar que la mayoría de proyectos estudiados usaron el primer método aquí explicado.

3.2 Configuración del *ARDUINO*

En lo que se refiere a la configuración del *ARDUINO,* se ha optado por utilizar un controlador *ARDUINO UNO,* debido a que el número de pines digitales y de pwm (*pulse width modulation)* que se requieren nos es suficiente. Lo que respeta a su velocidad y memoria también se adaptan a nuestras necesidades de uso.

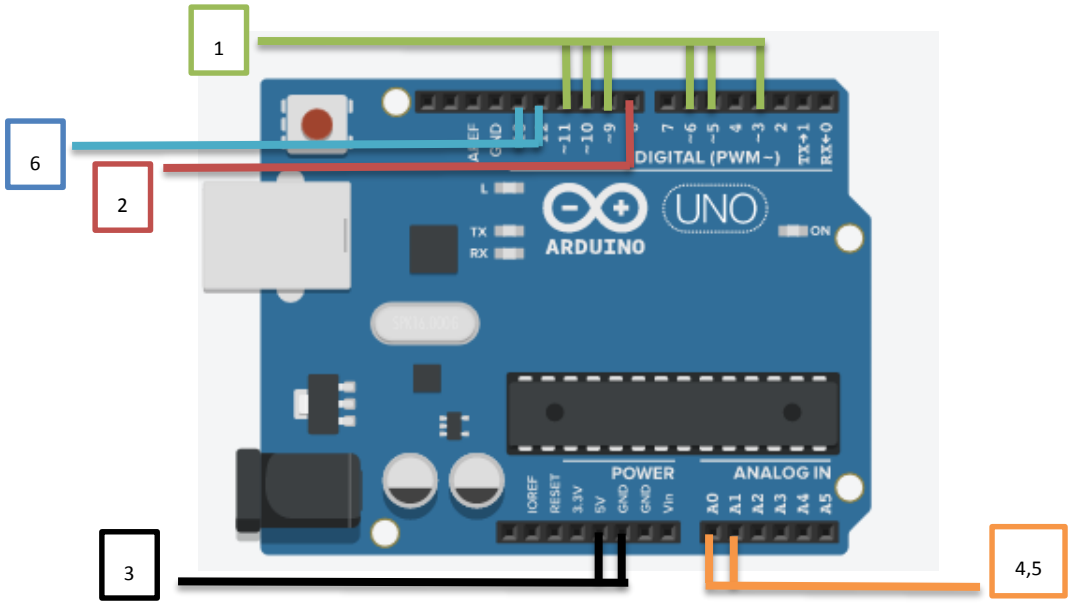


Figura 22 Conexionado general del controlador ARDUINO (Fuente: propia)

A continuación, se describen cada una de las conexiones mostradas y numeradas en la figura anterior:

1- Se dirigen a los controladores de los motores *bts7960b* , más en concreto a sus pines *pwm.* Por estos pines se transmitirán las órdenes de regulación para el control de velocidad de los motores.

2- Se dirige al *IRF 520 N*, este hará la misma función en el electroimán, que la que se realiza en los motores anteriormente mencionados.

3- Alimentación (Vcc y GND) de 5 voltios de los elementos de control. Tendrá una regleta bastante grande ya que deberá incluir la alimentación de todos los accesorios del sumergible.

4- El sensor de inundación utiliza una señal analógica para transmitir sus valores al controlador.

5- Así mismo, el sensor de presión diferencial también utiliza una señal analógica para, mediante su diafragma, transmitir la presión.

6- El módulo *MPU 6050* transmite los valores de la aceleración y el valor giroscópico por estos pines digitales.

Lo que se refiere al control y potencia de los motores y electroimán vamos a esquematizar el conexionado de la siguiente forma:

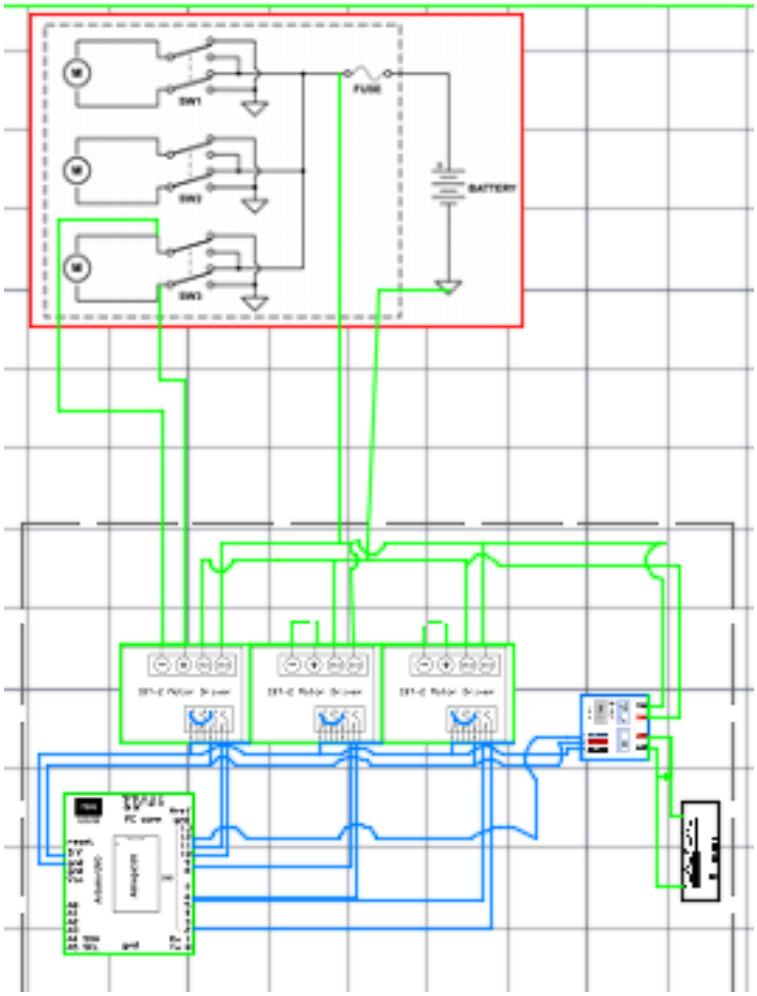


Figura 23 esquema eléctrico de potencia (verde) y control (azul) de la propulsión y electroimán (Fuente: propia)

3.3 *MATLAB*

Finalmente se seleccionó el *software* de *MATLAB* puesto que nos proporcionaba las herramientas necesarias para desarrollar aplicaciones de usuario de manera rápida e intuitiva. Aun así, hay otras opciones como es *LABVIEW* que también se adapta al *hardware* utilizado y permite realizar las mismas tareas.

El primer paso fue crear una interfaz gráfica "*GUI"* en *MATLAB* e ir introduciendo todos los accesorios que darán forma a nuestra interfaz gráfica (actuadores, textos estáticos, gráficas, casillas de chequeo…).

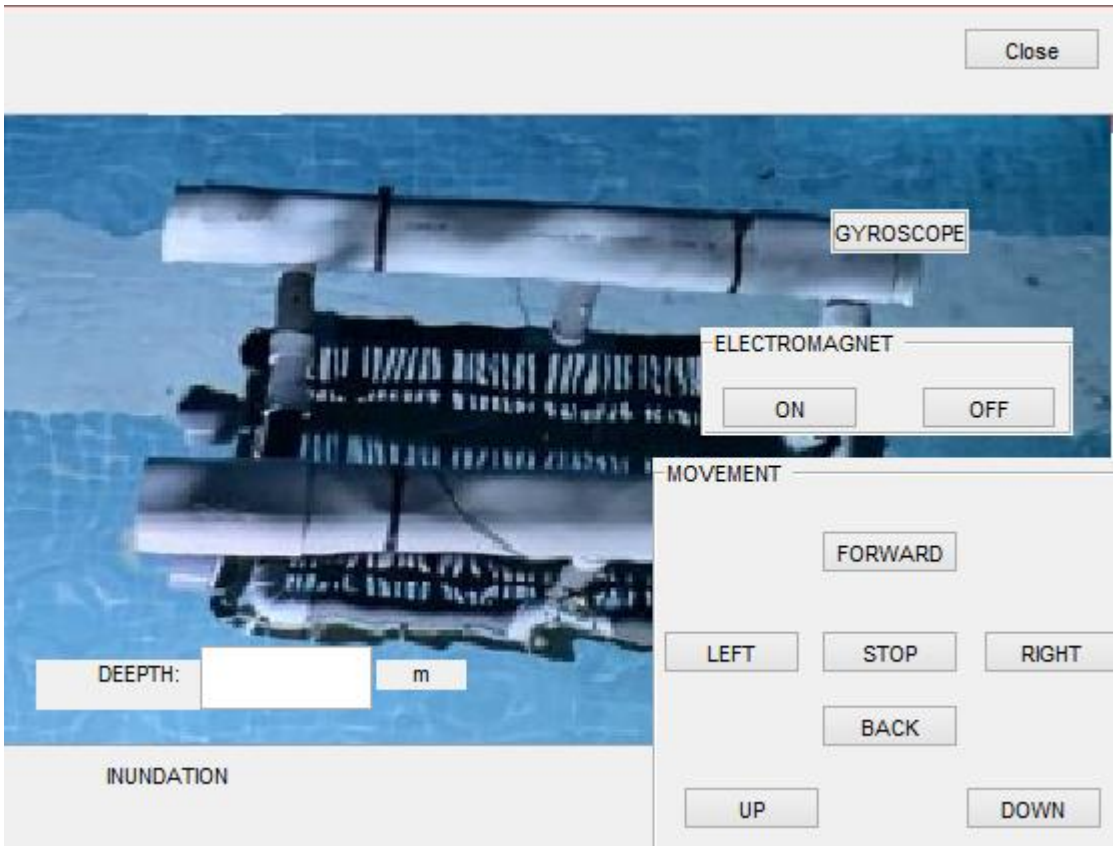


Figura 24 Interfaz gráfica

Teniendo estas casillas insertadas en nuestra interfaz debemos entrar en sus funciones *callback* para así poder programar cada una de sus funciones. Luego nos dirigimos a la función de variables de salida "*varargout "e* introducimos las variables globales, indicamos a *MATLAB* que comunique con *ARDUINO* mediante el puerto serie COM3, y escribimos el código que no entra en ninguno de los *callbacks* anteriormente mencionados como es el caso de la medición del sensor de inundación, la cual debe realizarse periódicamente e independientemente de si nosotros como usuarios clicamos sobre alguna función de retorno *callback.*

```
%LEFT
function pushbutton4_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;

writeDigitalPin(a,'D5',0);
for s1=0:0.1:0.5
writePWMDutyCycle(a,'D3',s1);
pause(1);
end

for s1=0:0.1:0.5
writePWMDutyCycle(a,'D9',s1);
pause(1);
```

```
end

writeDigitalPin(a,'D6',0);
```

Contenido dentro del *callback* de uno de los pulsadores

```
function varargout = CONTROL_FINAL_GUIDE_v2_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

global a;
global s1;
global d1;
global aux;
global Vout;
instrfind;
a=arduino('COM3');
instrfind;
 aux=0;
for i= 0 : 100000
  %  k=readVoltage(a,'A3')
    pause(2);
    if readVoltage(a,'A3') <= 0.1
        set( handles.text7,'ForegroundColor','blue')
    else
        set( handles.text7,'ForegroundColor','red')
    end
    drawnow;

 aux =(readVoltage(a,'A0')*5.0/1023.0);
 Vout=aux/10;
 p = ( Vout / 0.0018*5 ) - (0.04 / 0.0018);
 h = ((p*1000)/(997*9.8))*100;
 set( handles.edit1,'String',num2str(p))

end
```

Contenido global dentro de la función *varargout*

Capítulo 4. Accesorios e instrumentación

Esta parte consiste seleccionar los accesorios necesarios para cumplir el objetivo de nuestro *ROUV.* Luego se explica ciertas características de los accesorios que son necesarias para su correcto funcionamiento.

4.1 Selección de accesorios

En un primer estudio se ha visto viable poner los siguientes accesorios a nuestro vehículo:

- Sensor de presión, a través del cual podremos obtener la profundidad. La siguiente figura muestra la relación entre la profundidad y la presión.

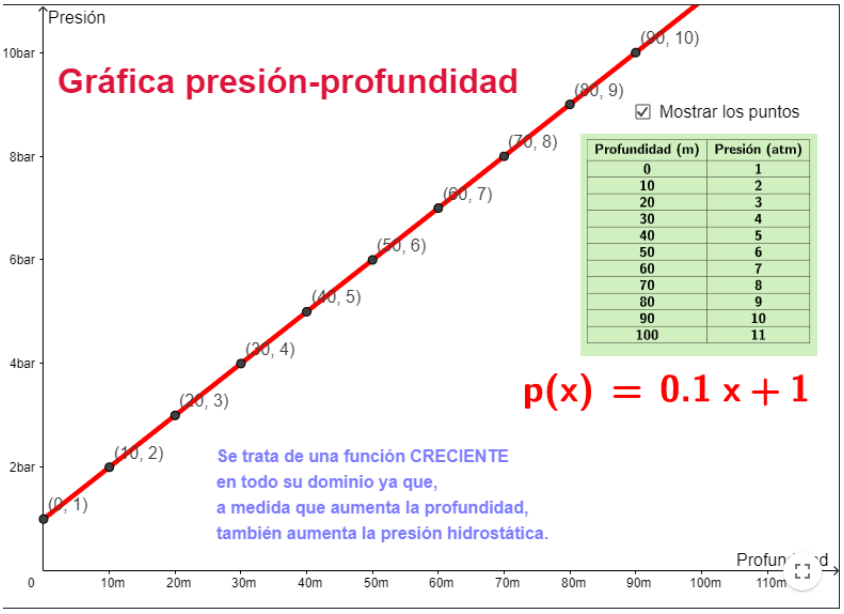


Figura 25 Relación presión-profundidad (Benito, 2020)

- Sensor de temperatura. Se descarta en principio debido a que en las profundidades donde se pretende trabajar (no más de 3 metros) hay variaciones muy pequeñas de la temperatura y el único fin que tendría el sensor sería el de registrar la temperatura en la *command box,* en particular la temperatura de los *motor drivers* ya que estos se calientan aun teniendo disipadores de calor y ese calor impide su correcto funcionamiento.

- Compás visual.Se escoge la opción del compás visual debido a que se producen campos electromagnéticos con toda la electrónica y sería imposible obtener un resultado coherente con un compás electrónico.Se usará un compás a modo de brújula y este será visto gracias al ojo de la cámara. También existe la opción de instalar un compás electrónico.

- Cámara

- Iluminación (luz de diodos + disipador de calor)

- Electroimán

- Magnetómetro

- Sensor de ultrasonidos. Como se va a implementar una cámara, no nos hará falta el sensor ya que veremos los posibles obstáculos con la cámara.

- Garra. En un principio se descarta y queda como una mejora del proyecto. Requiere de servos y programación para su funcionamiento.

- Acelerómetro/giroscopio

4.2 Características e implementación de los accesorios seleccionados

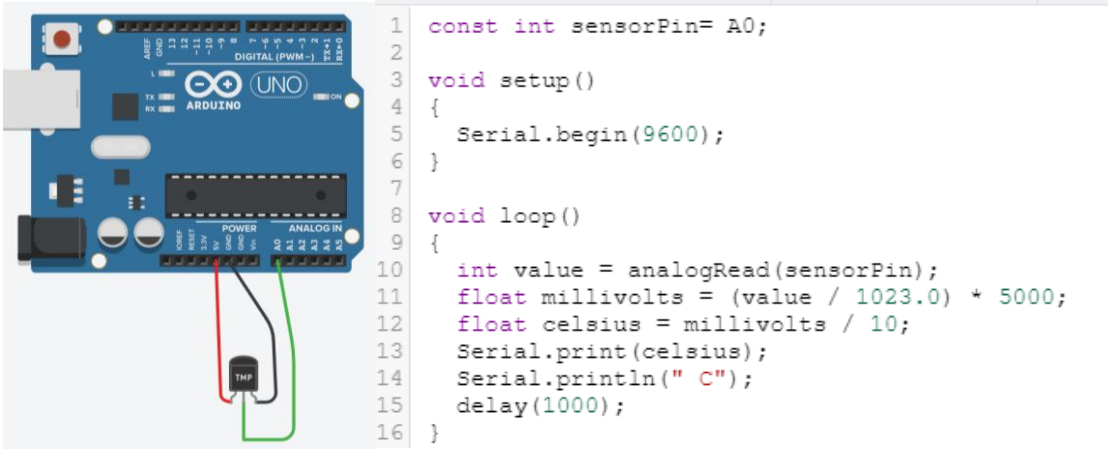
Después de realizar una búsqueda se ha decidido utilizar componentes educativos por su buena relación calidad-precio. A continuación, se describen sus principales características.

4.2.1 Sensores de temperatura

Con el fin de monitorizar la temperatura de los *motor driver*s se disponen unos sensores de temperatura TMP 35 de la casa ANALOG DEVICES, con la hoja de especificaciones técnicas incluida en la sección de referencias (DEVICES, 2020). Este sensor envía una señal analógica que es recogida por la unidad de control y mediante cálculos de *software* se consigue averiguar la temperatura.

Figura 26 TMP 35 (DIGCHIP, 2020)

A continuación, se muestra el conexionado individual de cada uno de estos sensores (3 en total) con nuestra placa *ARDUINO* así como el código que se implementará.



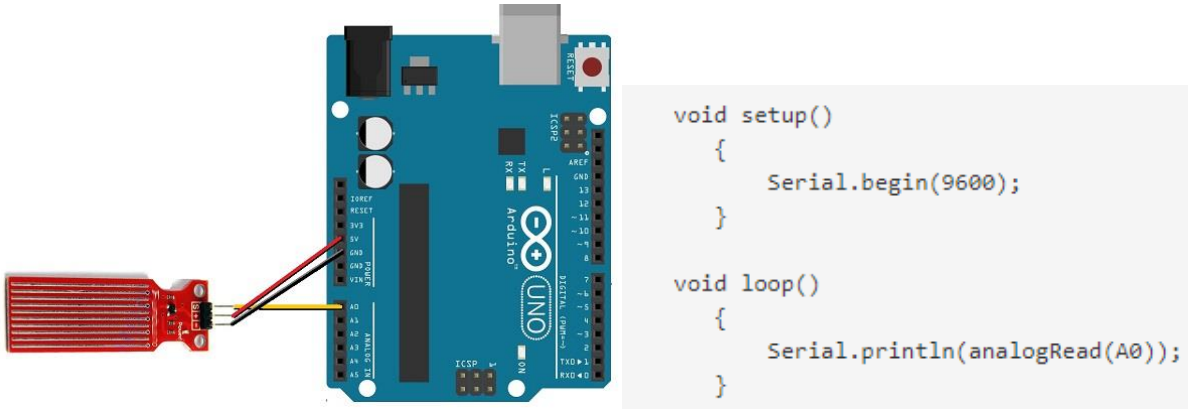
```
const int sensorPin= A0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int value = analogRead(sensorPin);
  float millivolts = (value / 1023.0) * 5000;
  float celsius = millivolts / 10;
  Serial.print(celsius);
  Serial.println(" C");
  delay(1000);
}
```

Figura 27 Montaje sensor-*ARDUINO* y código

Ayudarán a prevenir posibles fallos y deterioros de los controladores. Además, es un dispositivo muy simple y de muy fácil implementación. Se colocan en la zona del disipador de calor del controlador. Aunque finalmente, gracias a conseguir unos controladores muy potentes los cuales no tienen sobrecalentamiento en nuestras condiciones, su utilidad queda en segundo plano y quedan como posibles mejoras futuras.

4.2.2 Sensor de agua

Se utiliza el sensor de agua para comprobar que no entre agua en la *payload box.* Este sensor utiliza unos hilos conductores para, mediante una señal analógica, transmitir el nivel de agua que tiene la *payload box* gracias a la conductividad entre esos hilos. En el caso de que se detecte agua, el *software* hará saltar la alarma de inundación.



```
void setup()
   {
       Serial.begin(9600);
   }

void loop()
   {
       Serial.println(analogRead(A0));
   }
```

Figura 28 Montaje sensor-*Arduino* y código

4.2.3 Electroimán

Se utiliza un electroimán de 12 voltios y 0,8 Amperios que para su accionamiento se conectará a un módulo *IRF 520N*. Este es un transistor *MOSFET* introducido en un módulo que nos permite activar y desactivar el electroimán. Como ventajas respecto a la utilización de un relé convencional podemos decir que permite realizar una regulación *pwm*, la cual no será necesaria en nuestro caso.

Como puede apreciarse, al ser el electroimán una carga inductiva se debe añadir un diodo *fly-back* a modo de protección. Este diodo permite la recirculación de la carga inductiva cuando no se dé corriente y así se evita dañar el transistor o la placa *ARDUINO*.

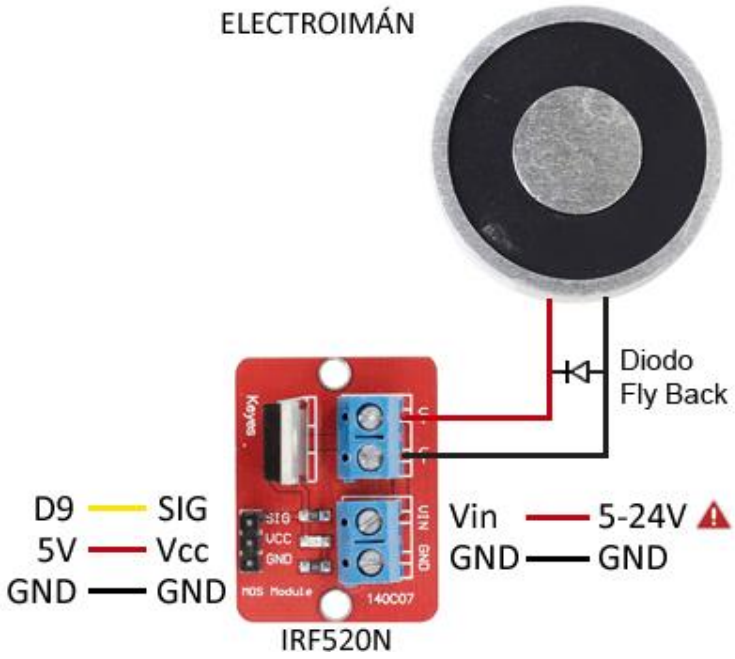


Figura 29 Configuración del electroimán con nuestro *Arduino* (LLamas, 2020)

4.2.4 Sensor de presión MPX

El sensor funciona gracias a 2 tomas conectadas a un diafragma cada una. Luego se mide la diferencia entre la presión de los 2 diafragmas para calcular la presión final.

Tiene una conexión a un puerto analógico de *ARDUINO* y los 2 pines de la alimentación. Los demás pines no se conectarán como puede apreciarse en la Figura 30. Una característica interesante de este sensor en concreto es que ya lleva un amplificador de señal incorporado sin él, el sensor emite a 40mV, con que el controlador no podría interpretar correctamente la señal.

Para calibrar el sensor, hay que programar en *MATLAB* una función que transforme el valor de la señal analógica (0-1023bits) que transmite el sensor mediante *ARDUINO* a unidades de presión.

Al utilizar finalmente un modelo *MPX 2200 DP, de* su *datasheet* obtenemos que dicha fórmula corresponde a:

A continuación, se añade a la fórmula un valor que reduce el error base del sensor. Este también se encuentra en el *datasheet* de nuestro sensor. Luego mediante pruebas, se realiza una regresión lineal y se introduce la ecuación de la recta en nuestro programa para obtener unos valores de presión más suavizados.

Para acabar transformamos el valor de la presión a altura o profundidad.

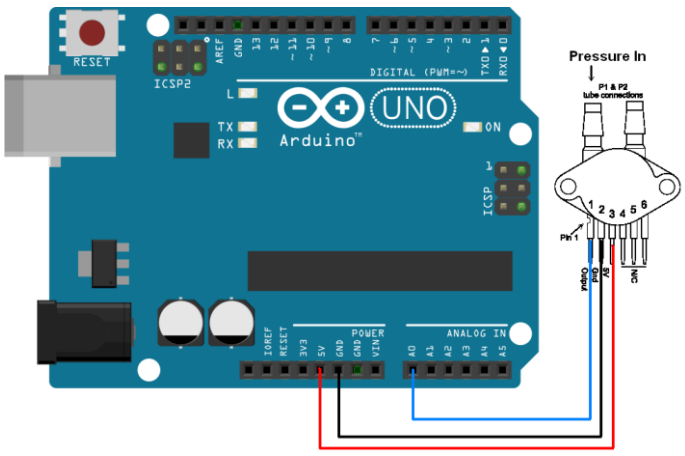


Figura 30 Montaje del MPX con *Arduino*

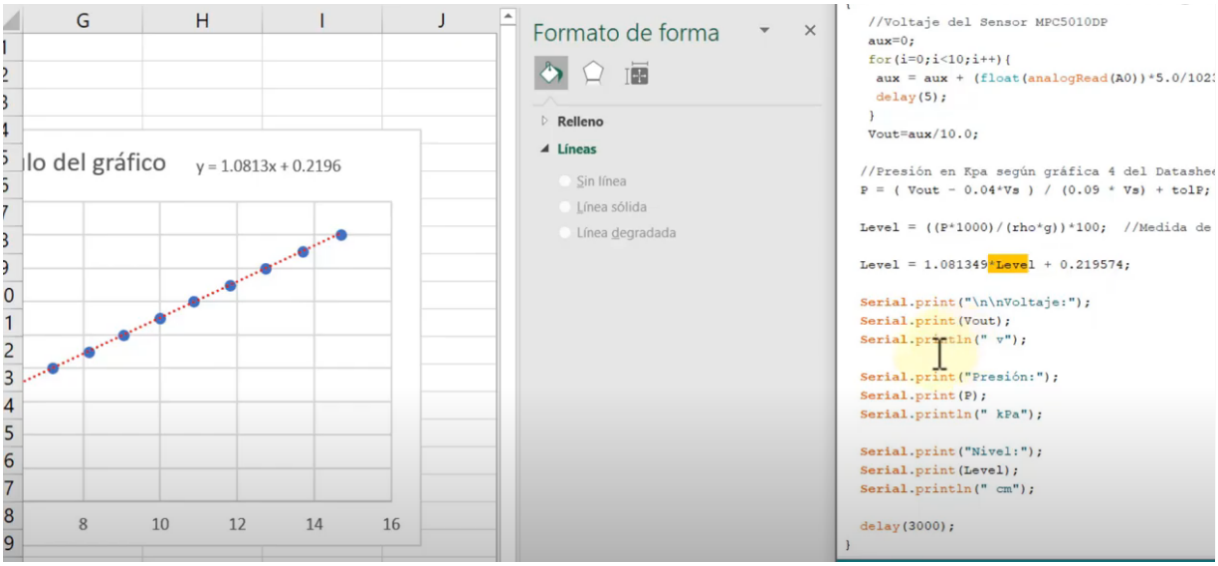


Figura 31 Regresión lineal introducida en el programa de *Arduino*

4.2.5 Acelerómetro-giroscopio MPU 6050

Se trata de una unidad *IMU (*unidad de medición inercial) que, mediante protocolo de comunicación I2C transmite los valores de aceleraciones y velocidades de los 3 ejes x,y,z. Como puede apreciarse en la figura, utilizamos 1 pin digital, los pines de alimentación (la unidad debe ser alimentada a 3,3V para su correcto funcionamiento) y los 2 pines que se encargan de realizar la comunicación con *ARDUINO.*

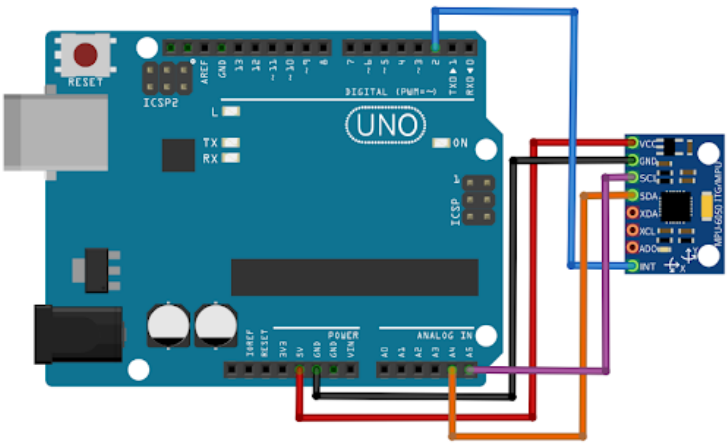


Figura 32 Conexionado del MPU 6050

La implementación se realizó de la siguiente forma:

1. Tras su correcta conexión se ejecutó, en la plataforma de *ARDUINO* el programa (JROWBERG, 2020) que realiza la comunicación giroscopio-*arduino* y transforma los valores recibidos en valores angulares.

2. Tras comprobar que los valores recibidos son correctos, se procede a utilizar el *sketch* de calibración (Code-blender, 2020) para obtener unos resultados correctos.

3. Luego, a la vez que se hace funcionar la *IDE* de *ARDUINO*, se conecta *MATLAB* para así poder realizar una figura intuitiva de los valores de escora de nuestro sumergible.
   En *MATLAB* programamos una gráfica y en ella delimitamos un prisma rectangular que hará de nuestro submarino.
   Finalmente realizamos la comunicación serial *ARDUINO-MATLAB* y podemos ver el resultado.

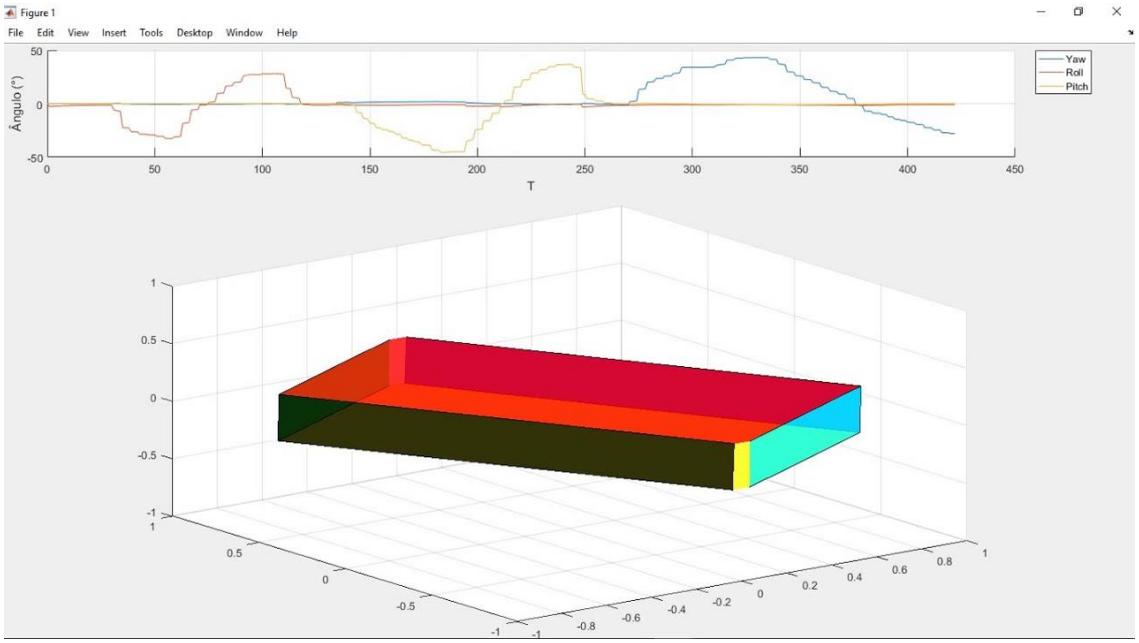


Figura 33 Figura de la interfaz gráfica final MPU 6050

Este componente, al ser de una complejidad considerable, dio bastantes problemas:

1. Aunque el módulo lleva un componente que reduce de 5 V a 3,3V este puede fallar y se puede acabar quemando el *MPU.* Esto es lo que nos pasó con uno de los ejemplares. Aparte de no transmitir ni recibir datos, se aprecia un ascenso considerable de la temperatura del chip cuando este está conectado.

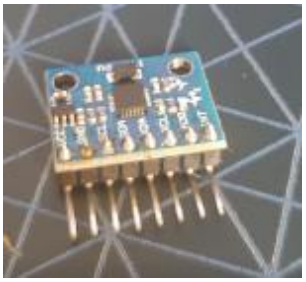


Figura 34 Si se observa detenidamente, puede apreciarse que el pin 2 está ligeramente quemado

2. Como ya se ha resumido anteriormente, su programación no es trivial y requiere varios pasos para poder obtener un resultado en ángulos y con la representación 3d correspondiente. Se tuvo problemas en programarlo con *MATLAB* para que pudiera salir en la interfaz gráfica junto a los demás sensores y componentes. Los problemas eran que utilizamos una variable (a) y su función era arduino, pero no permitía trabajar con dicha función el sensor y se tuvo que cambia por la función serial.

4.2.6 Cámara a tiempo real

Uno de los elementos indispensables con el que debería contar el sumergible es con una cámara que retransmitiese la imagen a tiempo real. Tras estudiar varias opciones se barajaron 2 opciones:

1. Una cámara *Arducam* o similar, que gracias a estar conectada con nuestro microcontrolador *ARDUINO* pudiera retransmitir la imagen. Esta, debería ser programada y transmite mediante el protocolo *SPI* (comunicación serial con periféricos) y utilizar el bus de transmisión I2C.

   Sus ventajas:
   - Pueden ser reprogramadas

   Sus inconvenientes:
   - Coste más elevado.
   - Muy baja calidad de imagen.
   - Posibles fallos y averías (tanto el *hardware* como el *software* son muy complejos).

2. Un equipo independiente, con su propia alimentación y pantalla LCD.

   Sus ventajas:
   - Al ser un equipo independiente el diagnostico de fallos es más sencillo.
   - El protocolo de transmisión de video e imagen está más especializado y permite una calidad mucho mayor.

- El costo es inferior.

Sus inconvenientes:

- No es moldeable a diferentes situaciones

Finalmente, las ventajas e inconvenientes hicieron que nos decidiéramos por la opción de utilizar un equipo independiente. La razón principal fue la calidad de imagen y video que recibiríamos con un equipo respecto al otro.

Capítulo 5. Montaje

En este apartado se encuentra la mayoría de la parte práctica del trabajo con lo cual se dividirá en sub apartados que harán el proyecto más pautado.

Se explica paso por paso como fue montado el *ROUV* desde el montaje del cuerpo exterior hasta el montaje y sellado de la *payload box.*Todo esto pasando por el añadido de motores, accesorios y montaje de la *command box.*

5.1 Cuerpo exterior (casco y apéndices)

Se explica paso por paso como fue montado el *ROUV* desde el montaje del cuerpo exterior hasta el montaje y sellado de la *payload box.*

Se ha optado por usar el *PVC* (poli cloruro de vinilo) para construir el esqueleto del vehículo. Teniendo ya las formas y dimensiones solo nos falta comprar tubos de *PVC* de las secciones y largura necesaria junto a sus codos correspondientes. Luego se procede a cortarlos y juntar el conjunto de piezas.

Al cortar los tubos hay que tener en cuenta que los diferentes codos añaden una longitud considerable al tubo final.

A continuación, añadiremos una plataforma en la base del vehículo para poder disponer sobre de ella la *payload box*, motores y ciertos accesorios. En mi caso se ha seleccionado la base de una caja de fruta para hacer de función.

Luego procedemos a juntar todo el conjunto, añadimos soldador de PVC para conseguir unir el conjunto y finalmente taladramos ciertos agujeros en sitios estratégicos para que la estructura principal se llene de agua.

Ahora, mediante bridas juntamos los tanques de lastre inferiores y los tanques de aire a la estructura principal.

Figura 35 Estructura con base

El siguiente paso fue acoplar a la estructura, los tanques de lastre. Se ha optado por la distribución clásica en estos sumerguibles, la cual consiste en disponer dos grandes tanques sellados y llenos de aire en la parte superior y dos tanques inferiores de menor tamaño que llevarán el lastre indicado para, tras el cálculo y pruebas de flotabilidad conseguir que esta sea neutra.

Figura 36 Vista desde popa del *ROUV*

Para el lastrado se utilizaron barras de refuerzo de construcción cortadas. Estas entran justo en los tubos y no permiten ningún movimiento que pueda desestabilizar el sumergible. La cantidad de kilogramos de material usado corresponde a la calculada en 2.3.2 Análisis de flotabilidad

Figura 37 Barras de metal a modo de lastre

Finalmente, tras comprobar que todo está fijo, se procede al pintado del sumergible. Se ha escogido una pintura anticorrosiva, en spray y de color blanco nuclear que pueda aguantar el agua y sea visible.

Figura 38 Acabado tras el pintado

5.2 Motores

Se han seleccionado unas bombas de achique para hacer la función de motores. Su principal ventaja es que no hace falta garantizar su estanqueidad ya que ya son estancas de por sí.

Luego se unen los motores al cuerpo exterior mediante bridas de fijación y se les insertan las hélices. En nuestro caso hemos escogido unas hélices de 3 palas.

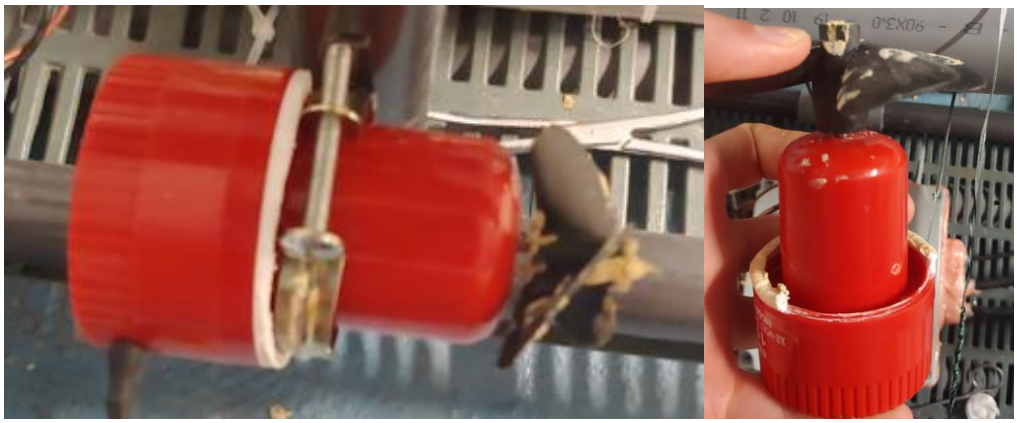


Figura 39 de izquierda a derecha Propulsor horizontal y vertical

5.3 Accesorios

5.3.1 Electroimán

Se ha alojado en el fondo de la caja estanca *payloadbox.* Con esta disposición evitamos tener que hacer pasar sus cables a través de la caja y obtenemos un vehículo más compacto.

Al crear un campo electro-magnético con corriente continua, no tendremos problemas con los demás sensores y controladores que se alojan dentro de la caja estanca. El único inconveniente que se puede sacar de esta disposición es que el electroimán se calienta bastante, pero al ser de uso muy puntual podemos asumirlo.

5.3.2 Sensor de inundación

Se dispuso de forma vertical con que, mediante conductividad eléctrica, nos indicará el nivel de agua que hay en la caja estanca.

5.3.3 Sensor de presión MPX

Uno de los diafragmas se conectó con un tubo de material plástico al exterior de la caja estanca. El otro medirá dentro de la caja, para luego midiendo la diferencia saber la presión a la que nos encontramos.

A continuación, se procedió a dejar unos centímetros cúbicos de colchón de aire entre el diafragma y el agua. Esto permitirá que la vida del sensor se prolongue considerablemente.

5.3.4 Sensor IMU

Aparte del conexionado, asegurarse de que el sensor queda fijado en posición completamente horizontal al sumergible. En nuestro caso se realizó fijándolo con tornillos y tuercas.

5.3.5 Cámara a tiempo real

Esta fue dispuesta en la proa del vehículo y sujetada mediante bridas de fijación. Su junta con el cable es estanca.

Por otro lado, la pantalla que usaremos para ver la imagen se fijará en el mando de control. Así, aunque no dispongamos de ordenador podemos operar el vehículo y realizar una inspección visual de nitidez.

5.4 Command box

Es el medio de control directo de la propulsión del vehículo, además de disponer de los adaptadores necesarios para poder realizar la comunicación serial entre el microcontrolador y el programa *MATLAB.*

El primer paso es perforar la *command box* seleccionada e introducir en ella los interruptores *dpdt (*de doble polo y doble reversión).

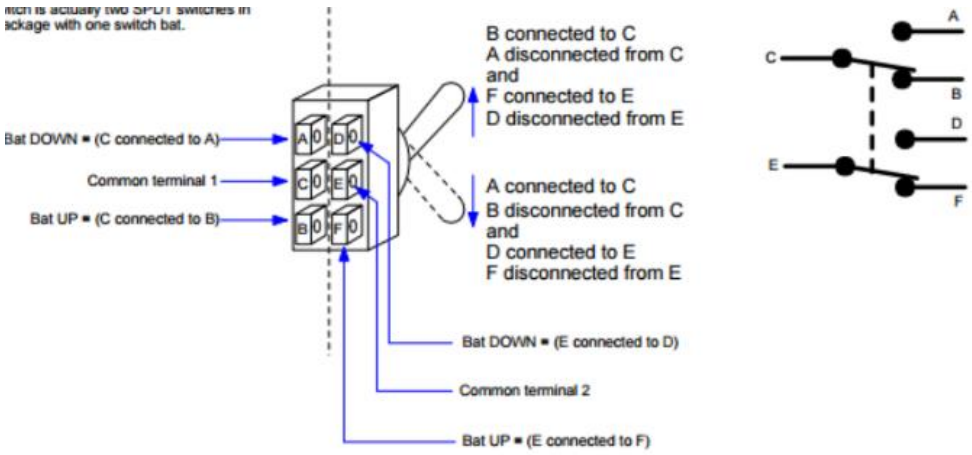


Figura 40 Tipo de interruptores utilizados para el control mecánico (UPC, 2020)

A continuación, repetir el paso anterior, pero para poder introducir el pasar el cable USB y los cables de alimentación.

Finalmente cablear toda la *command box* para que esta quede operativa. Además, en ella irá un fusible de 10 A conectado a la fuente de alimentación a modo de elemento de seguridad para evitar sobrecargas de los diferentes elementos del equipo.

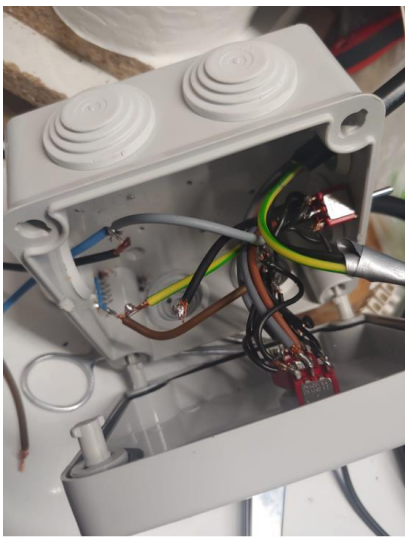


Figura 41 Soldando cableado de potencia para los interruptores

Figura 42 *Command box* acabada

El motivo de usar un fusible de 10 Amperios en la línea de potencia general es el de proteger de cortocircuitos a los equipos. En nuestro caso, los consumidores de potencia requieren un amperaje que se ajusta a esos 10 Amperios. Consumir más quemaría el fusible y habría que revisar la instalación y cambiar el fusible.

En el ANEXO se puede apreciar la disposición final de todos los elementos que componen la *command box.*

5.5 Payload box y sellado

Es el elemento que contiene toda la electrónica de control y gran parte de la que se utiliza para la potencia. Es por eso que la organización, separación y fijación de los elementos es fundamental. En nuestro caso, la caja estanca IP 67 escogida ha sido capaz de albergar a todo el equipamiento.

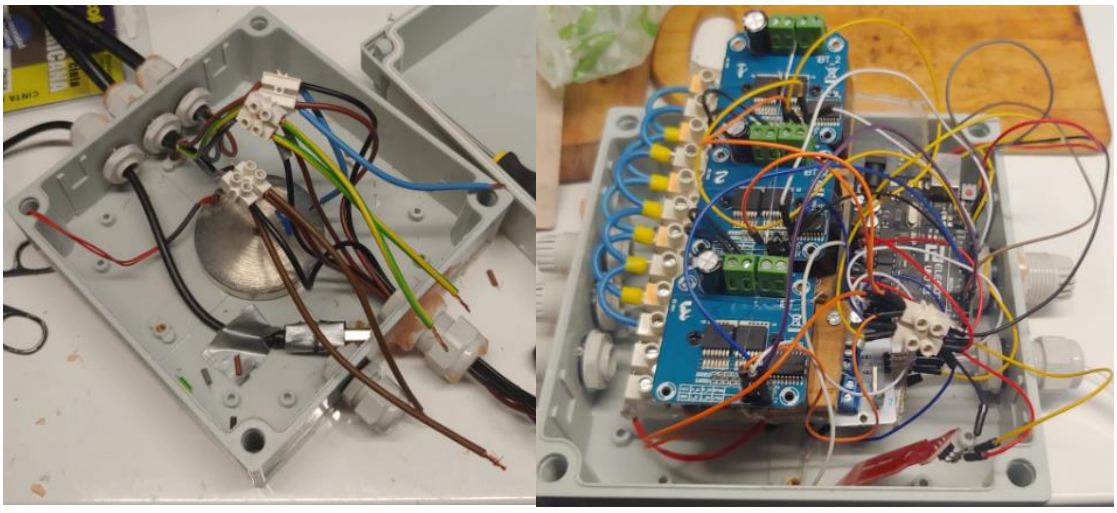


Figura 43 Disposición de la *payload box* (izquierda corresponde a potencia y derecha control)

Después de comprobar que todo cabe, se ha sacado la electrónica y se han empezado a hacer los agujeros que permiten a los cables acceder a la caja estanca. El sello se conseguirá mediante prensaestopas, juntas tóricas y cera o silicona.

A continuación, se cortó un trozo de cable, se pasó por los prensaestopas como si fuera el cable definitivo y se selló la caja (en nuestro caso utilizamos cera). El siguiente paso fue probar la estanqueidad debajo del agua por un periodo prolongado.

El resultado fue exitoso. Se comprobó la estanqueidad por un periodo de 30 minutos a 1 metro de profundidad.

Figura 44 Entrada del cableado a la caja estanca

Seguidamente se procedió a introducir todo el contenido dentro de la caja y antes de cerrarla, probar que todo el conjunto funcione, tanto mecánicamente (mediante interruptores) como de forma digital (gracias a la comunicación serial y la interfaz gráfica del programa *MATLAB).*

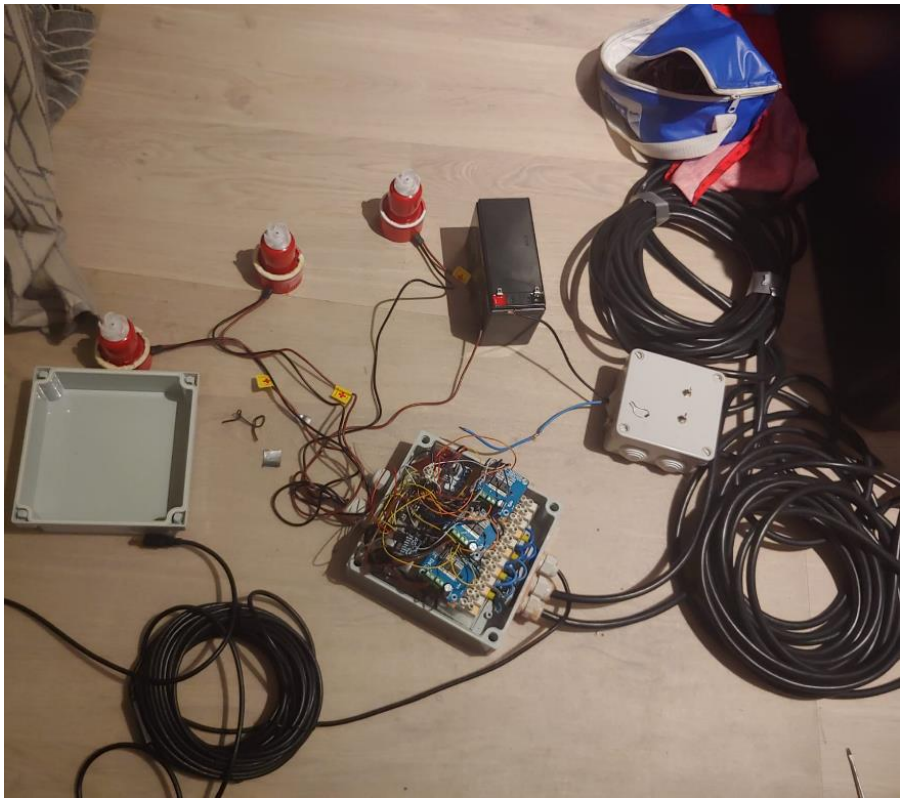


Figura 45 Conjunto de electrónica que compone el *ROUV*

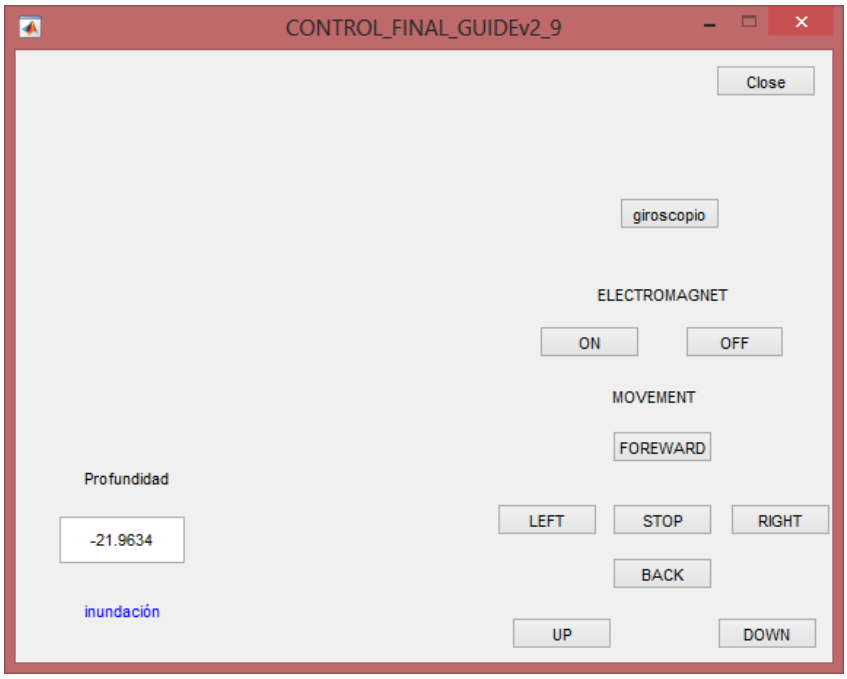


Figura 46 Prueba de la interfaz gráfica del usuario

Los resultados fueron satisfactorios. Una vez adaptada la unidad *IMU* y calibrado el sensor de presión funciona correctamente. En el ANEXO se puede apreciar la disposición final de todos los elementos que componen la *payload box.*

5.6 Umbilical y transmisión de datos

De primeras se iba a utilizar la interfaz RS 232 para la transmisión de datos entre el PC y el microcontrolador *ARDUINO.* Tras investigar se consiguió utilizar comunicación mediante USB (el método estándar que utiliza el microcontrolador). Eso es posible ya que a día de hoy se puede alargar un cable USB hasta nuestros requeridos 15 metros gracias a la utilización de un repetidor de señal en medio del cable.

Figura 47 Repetidor en cable USB (DELEYCON, 2020)

Como cabe de esperar, este repetidor será sellado para su uso debajo del agua. El sello se realiza bañándolo en cera.

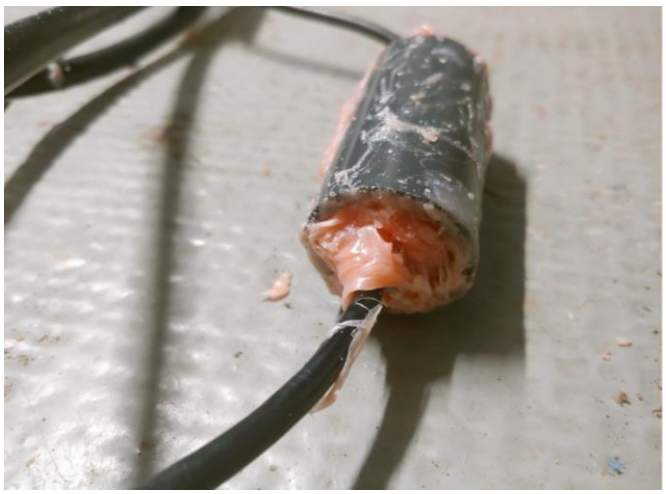


Figura 48 Repetidor encerado y sellado con silicona

Lo que a la transmisión de datos respecta es simple, se utiliza una comunicación serial (es decir, se manda y recibe datos de uno en uno), lo cual para nuestro proyecto es aceptable ya que las mediciones de sensores que tenemos nos permiten ese minúsculo retraso entre una comunicación y la otra.

A continuación, unimos todos los cables con manguera y espuma para que el cable no se hunda y afecte a la estabilidad del vehículo.

Figura 49 Proceso de unión del umbilical

Capítulo 6. Pruebas experimentales

En este capítulo se presentan y analizan los resultados de las pruebas experimentales. Se han llevado a cabo dos tipos de experimentos a fin de analizar el comportamiento estático y el comportamiento dinámico.

6.1 Comportamiento estático y flotabilidad

En primera instancia se realiza la prueba estática en la cual se quitan todos los elementos que puedan estropearse debido a una entrada de agua en la *payload box* y se sella esta como si el proyecto estuviera acabado. Luego se sumerge el *ROUV* manualmente y se comprueba que la estabilidad sea la correcta. Al sacar el artefacto se comprueba que no hay entrada de agua.

Es hora de probar la estabilidad y flotabilidad neutra del aparato. Después de modificar lastre, el resultado fue exitoso.

Figura 50 Primera prueba de flotabilidad y estabilidad sin electrónica

6.2 Comportamiento dinámico y final

La prueba final es la dinámica, en la cual se reinsertan todos los componentes electrónicos y se sumerge el *ROUV* esta vez ya a distancia. Se hace una inmersión en espiral para comprobar y/o modificar la flotabilidad.

Figura 51 Preparación y para la prueba dinámica

La primera inmersión resultó satisfactoria. La flotabilidad era ligeramente positiva y el funcionamiento del accionamiento mediante interruptores con el mando también. Se pudo apreciar que los motores no tenían problema para desplazar al gran sumergible pese a los pequeños propulsores instalados en los motores horizontales.

Figura 52 Sumergible utilizando el desplazamiento horizontal

La siguiente parte de la prueba es probar el control y monitorización de datos utilizando la interfaz gráfica en *MATLAB.* Se procede a conectar el *USB a* un ordenador con el *software* de *MATLAB* instalado y realizar la prueba. E aquí donde puede apreciarse el arranque suave de los motores utilizando el *pulse width modulation* y la recepción y transmisión de datos. En esta prueba bajamos el *ROUV a* una profundidad considerable y calibramos el sensor de presión *MPX.*

Se realiza una espiral descendente y luego un ascenso recto simulando un caso de emergencia. El sumergible se comporta satisfactoriamente. El giro horizontal completo se realiza en apenas un par de segundos y el desplazamiento horizontal se adapta a los 1,5 nudos impuestos anteriormente. El umbilical se hunde con que podríamos aumentar la cantidad de espuma para mejorar su flotabilidad y la cámara ha dado un resultado óptimo, una imagen nítida y luminosa gracias a los *LEDS* incorporados.

Capítulo 7. Presupuesto

A continuación, se presenta todo el material del cual está compuesto el ROUV

Nombre	Cantidad	Precio (€)
Componentes *PVC:*		
Tubo D90 1M	1	3.09
Tubo D50 2M	1	3.49
Tubo D32 5M	1	5.38
Manguitos D90	3	2.7x3
Tapón 50MM Hembra	4	1.69x4
Injerto D32	12	0.56X12
Codo D32	8	0.22X8
Bomba achique 12V	3	24.99x3
Caja estanca 100x100	1	1.99
Sensor de presión MPX2200DP	1	19.94
Conmutador DPCO	3	2.92X3
Portafusiles	1	0.85
Conector DB-9 hembra	1	0.36
Prensaestopas 16 mm	5	0.59x5
Cable RVK 25 m	1	19
T métrica	1	1.4
Prensaestopas 20 mm	1	0.72
Juntas goma 3/4	1	1.89
Juntas goma racor plano	1	1.98
Abrazadera metálica pack	1	3.3
Tuerca hexagonal d.4	1	1.56
Adhesivo soldador tuberías	1	3.22

Spray ANTIÓX	1	4.95
Regleta 12 p	1	1.39
Cámara y focos	1	58.29
Caja estanca IP 67	1	13.33
Módulo amplificador Op.	1	0.26
Motor driver BTS 7960	3	3.18
Electroimán	1	2.96
Modulo Sensor Nivel de Agua	1	1.8
Sensores de temperatura	3	6.15
IMU MPU 6050	1	2.75
Controlador L298N	3	8.16
USB 15 M	1	19.99
Hélice 3 palas	3	2.79x3
Mircocontrolador Arduino UNO	1	10
Gastos energéticos (1 KWh=0,1€)	Aprox 500	50
TOTAL:		378.68 €
Horas de trabajo	1080	N/A

Capítulo 8. Conclusiones y líneas futuras

Se ha diseñado y construido un prototipo de vehículo de inspección (de los cascos de buques, pantalanes y en general de medias aguas y el fondo marino), manipulación de objetos ferromagnéticos , toma y muestra de ciertos parámetros. Todo ello siguiendo un plan de trabajo marcado por los objetivos, plazos de entrega y limitaciones.

En lo que respecta a los plazos de entrega se pretendía realizar el proyecto en 8-9 meses de trabajo , aun así se ha sido flexible, sobre todo en lo que se refiere a la parte final y práctica ya que debido a la situación de pandemia, la llegada de las piezas se ha visto ligeramente afectada y la movilidad reducida con que la mayoría del proyecto recaía sobre una persona (aun así las reuniones a distancia han permitido avanzar en el proyecto y evitar posibles futuros errores) y los pequeños errores que han ido apareciendo han tenido que ser resueltos más adelante, desmontando lo montado y solucionándolos. En lo que hace referencia a la primera parte de búsqueda de información esta ha sido muy fructífera ya que existe mucha información en la red y muchos proyectos de robots muy bien explicados. De allí que he decidido realizar esta memoria en formato manual o libro para que otras personas interesadas, puedan reproducir y mejorar mi proyecto.

Las limitaciones y restricciones con las que partíamos eran temporales, económicas; se ha podido realizar el proyecto utilizando material educativo y de bajo coste sin tener una gran repercusión en la calidad del prototipo. Si quisiéramos hacer un salto en la calidad del proyecto nos supondría un coste mucho mayor al de calidad/precio obtenido actualmente. También, al iniciar el proyecto planteamos unas restricciones funcionales para encaminar las características del *ROUV.* Estas se han cumplido casi en su totalidad destacando la longitud de alcance (15 metros), sensores implementados, profundidad de inmersión. Las restricciones de dimensión han tenido que ser ligeramente modificadas, algunos sensores se han dejado para futuras mejoras, la cámara ha sido independiente tras sopesar las ventajas e inconvenientes respecto a una cámara integrada en el sistema y la estanqueidad total ha supuesto un problema hasta que se ha decidido implementar cera y silicona para sellar las juntas.

¿Qué hay de novedoso y original en el proyecto? La manipulación de objetos debajo del agua vía electroimán es un punto original y práctico ya que, el imán no ocupa lugar (como una garra móvil) y no estorba (puede activarse y desactivarse) y está ubicado en un sitio donde el objeto recogido no compromete a la estabilidad del sumergible. Por otro lado, cabe destacar que es un vehículo de unas dimensiones considerables (respecto a otros vehículos de inspección superficiales) lo que le da ventajas de poder realizar *upgrades* e introducir nuevos componentes como sensores, garra, cámaras laterales…

Los controladores de los motores, que normalmente sufren de sobrecalentamiento y fallan, han sido sobredimensionados con lo cual el problema del fallo por exceso de calor ha sido solucionado.

Otro punto a remarcar es que se ha conseguido que la interfaz gráfica sea un programa intuitivo y fácil de usar para el operario, además de que si no se dispone de ordenador o el *software* necesario no se tiene disponible puede operarse el sumergible y ver su imagen gracias a un mando de control mecánico.

Los componentes que mejor resultado han dado han sido la cámara a tiempo real, el sensor de inundación y el sensor de presión *MPX.* En cambio se ha tenido problemas al introducir en el mismo programa la unidad inercial, además de que es muy sensible y una fue quemada por los 5 voltios de *ARDUINO* y el electroimán debido a un sobrecalentamiento y falta de poder de magnetización al activarlo (tras estudiarlo se ha llegado a la conclusión de que el fallo ocurre debido a que la batería utilizada para alimentar al equipo apenas tiene suficiente amperaje para manejar los motores, con que si se utiliza una batería de mayor amperaje, el problema quedaría resuelto).

Como líneas futuras se puede realizar el cambio de interfaz gráfica *GUIDE* a *APP-DESIGNER*. Esta última es más moderna y tiene unas cuantas opciones más. El proceso de traspaso de código de una a otra es simple ya que ambas son del mismo *software.* El cableado del mando de control se puede mejorar ya que tiene algún que otro fallo tocándose dos cables de los interruptores y haciendo que estos fallen. Simplemente utilizar cable de 0,75 mm^2 en aluminio el cual es más manejable y permite adoptar posiciones más dobladas que el cable clásico de cobre. Plantear incorporar una garra movida con servomotores para manipular materiales no metálicos. Y añadir una gran variedad de sensores, mencionados en 4.1 Selección de accesorios.

En pocas palabras ha sido un proyecto en el cual he podido aplicar conocimientos y aprendido acerca de electrotecnia, programación, gestión de proyectos, teoría del buque e hidrodinámica. También a saber salir de apuros y problemas improvisando y buscando soluciones prácticas.

Referencias

&JunH, J. P. (18 de 4 de 2020). *semanticschoolar.* Obtenido de https://www.semanticscholar.org/paper/Revisiting-the-Challenger-Deep-Using-the-ROV-Kaiko-Barry-Hashimoto/1a09d25df08f5512220aecc3c45e7f9f70f589fd

Allmediguer, E. (2020). *Submersible vehicles system design.* SNAMR.

Anxel, H. y. (18 de 04 de 2020). *20minutos.* Obtenido de https://blogs.20minutos.es/trasdos/2015/08/03/cutaway-work-ilustracion-enciclopedica/

Arbeam. (18 de 04 de 2020). *World press.* Obtenido de https://bremolympicnlus.wordpress.com/2014/10/03/dsrv-1-mystic-installed-at-the-naval-undersea-museum-keyport/

B.Robert. (17 de 04 de 2020). *ABC.* Obtenido de https://www.abc.es/tecnologia/20140504/rc-inspeccion-robotica-submarina-201405041730.html?ref=https%3A%2F%2Fwww.google.es%2F

Benito, S. (17 de 2 de 2020). *geogebra*. Obtenido de https://www.geogebra.org/m/RnvdacGf

Buckley, I. (4 de 4 de 2020). *makeuseorf*. Obtenido de https://www.makeuseof.com/tag/best-microcontroller-boards/

Code-blender. (20 de 10 de 2020). Obtenido de https://codebender.cc/sketch:97892#MPU6050%20calibration.ino

DELEYCON. (6 de 10 de 2020). *Amazon*. Obtenido de https://www.amazon.es/deleyCON-Impresora-Amplificador-Repetidor-Computadora/dp/B07Q4QC7NH/ref=sr_1_6?adgrpid=56266996255&dchild=1&gclid=CjwKCAjwq_D7BRADEiwAVMDdHg9f9BI5TAJjbKz2NP9Qw27rpVcE6QFHXIYlceCmYVuxA6_oMMV8cBoCmLAQAvD_BwE&hvadid=275493534223&hvdev=c

DEVICES, A. (15 de 4 de 2020). *ALLDATASHEET.* Obtenido de https://www.alldatasheet.com/datasheet-pdf/pdf/49107/AD/TMP35.html

DIGCHIP. (15 de 04 de 2020). *DIGCHIP.* Obtenido de https://www.digchip.com/datasheets/8820330-tmp37ft9z-sensor-temp-anlg.html

Geirandersen. (15 de 2 de 2020). *https://www.robotshop.com*. Obtenido de https://www.robotshop.com/community/forum/t/myrov/13250

GRAVIT0N, S. y. (23 de 2 de 2020). *PIKABU*. Obtenido de https://pikabu.ru/story/pogruzhenie_na_samodelnoy_podvodnoy_lodke_6188508/author

homebuilrtovs. (15 de 2 de 2020). Obtenido de http://www.homebuiltrovs.com/seafox.html

JROWBERG. (22 de 10 de 2020). *GITHUB*. Obtenido de https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

Killer, E. (12 de 5 de 2020). *TCPalm.* Obtenido de https://eu.tcpalm.com/story/news/local/indian-river-county/2019/05/16/victor-vescovo-triton-sub-dives-mariana-trench-five-deeps-expedition/3665789002/

Leroy M. (12 de 5 de 2020). Obtenido de https://www.leroymerlin.es/fp/19792416/bomba-de-achique-jn-de-500-w-y-caudal-maximo-de-4200-l-h

LLamas, L. (13 de 8 de 2020). Obtenido de https://www.luisllamas.es/electroiman-arduino/

name, F. n.-l. (4 de 4 de 2020). *youtube*. Obtenido de https://www.youtube.com/watch?v=QNLpBRq5Gvo

NavSource. (18 de 4 de 2020). *NavSource.* Obtenido de http://www.navsource.org/archives/08/08442.htm

oceaneering. (18 de 4 de 2020). Obtenido de https://www.oceaneering.com/

OER. (12 de 5 de 2020). Obtenido de https://oceanexplorer.noaa.gov/technology/subs/hercules/hercules.html

Phatowary, K. (18 de 04 de 2020). *amusingplanet.* Obtenido de https://www.amusingplanet.com/2020/03/bathysphere-worlds-first-deep-sea.html

Proto57. (4 de 2 de 2020). *youtube*. Obtenido de https://www.youtube.com/watch?v=TfJ0Y3oZTzQ

Rebikoff-Niggler foundation. (18 de 04 de 2020). Obtenido de https://www.rebikoff.org/history/

ROIG, J. B. (4 de 4 de 2020). *UPC Commons*. Obtenido de https://upcommons.upc.edu/bitstream/handle/2117/81559/111193_TFG2015%20Joan%20Bauza%20Roig.pdf?sequence=1&isAllowed=y

Sanchez, B. (4 de 3 de 2020). *geogebra*. Obtenido de https://www.geogebra.org/m/JXZ8T7Bf

SIL. (26 de 10 de 2020). *CODEBENDER*. Obtenido de https://codebender.cc/sketch:97892#MPU6050%20calibration.ino

SpiceSHipOne. (4 de 2 de 2020). *Instructables*. Obtenido de https://www.instructables.com/id/Underwater-ROV/

UPC. (7 de 10 de 2020). *https://telecos.upc.edu/es*. Obtenido de https://telecos.upc.edu/es

vetus. (12 de 5 de 2020). Obtenido de https://www.vetus.com/media/magentominds/sasdocument/20151019132622_0.pdf

Weckhunter. (18 de 4 de 2020). Obtenido de https://wreckhunter.net/DataPages/thresher-dat.htm

ANEXO 1. Códigos adjuntos

1. Código de *ARDUINO*

Antes de todo, una breve mención a las librerías usadas en nuestro proyecto.

- *Robojax-BTS7960*, usada, en la propulsión para hacer funcionar a los *motor drivers* BTS.
- *DallasTemperature*, la cual como su nombre indica se utilizaría en los sensores de temperatura TMP.
-
- *One Wire* usada en la unidad inercial *MPU,* esta permite la conexión con ciertos sensores de 1 cable. Usada con la gran mayoría de sensores educacionales que dispone *ARDUINO* con que es una librería imprescindible.

Tras tener las librerías instaladas en el *IDE,* los códigos utilizados en *ARDUINO* son los siguientes.

1. Calibración del *MPU6050,* debemos antes de todo, calibrar los ejes del giroscopio para que este, cuando se encuentre en posición totalmente horizontal, no de valores de escora. Esto se consigue aplicando a *ARDUINO* el código de (SIL, 2020).

2. A continuación, tras calibrar el sensor podemos correr en nuestra *IDE* el código *MPU_DMP_6,* el cual ya viene en los ejemplos y nos permite obtener los valores de las velocidades y aceleraciones realizando el cambio de valores en bits a ángulos.

```
// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class using DMP (MotionApps v2.0)
// 6/21/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2013-05-08 - added seamless Fastwire support
//                - added note about gyro calibration
//     2012-06-21 - added note about Arduino 1.0.1 + Leonardo compatibility error
//     2012-06-20 - improved FIFO overflow handling and simplified read process
```

```
//     2012-06-19 - completely rearranged DMP initialization code and simplification
//     2012-06-13 - pull gyro and accel data from FIFO packet instead of reading directly
//     2012-06-09 - fix broken FIFO read sequence and change interrupt detection to RISING
//     2012-06-05 - add gravity-compensated initial reference frame acceleration output
//                - add 3D math helper file to DMP6 example sketch
//                - add Euler output and Yaw/Pitch/Roll output formats
//     2012-06-04 - remove accel offset clearing for better results (thanks Sungon Lee)
//     2012-06-01 - fixed gyro sensitivity to be 2000 deg/sec instead of 250
//     2012-05-30 - basic DMP initialization working

/* ============================================
I2Cdev device library code is placed under the MIT license
Copyright (c) 2012 Jeff Rowberg

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
===============================================
*/

// I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files
// for both classes must be in the include path of your project
#include "I2Cdev.h"

#include "MPU6050_6Axis_MotionApps20.h"
//#include "MPU6050.h" // not necessary if using MotionApps include file

// Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation
// is used in I2Cdev.h
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
```

```
    #include "Wire.h"
#endif

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;
//MPU6050 mpu(0x69); // <-- use for AD0 high

/* =========================================================================
   NOTE: In addition to connection 3.3v, GND, SDA, and SCL, this sketch
   depends on the MPU-6050's INT pin being connected to the Arduino's
   external interrupt #0 pin. On the Arduino Uno and Mega 2560, this is
   digital I/O pin 2.
 *  ========================================================================
*/

/* =========================================================================
   NOTE: Arduino v1.0.1 with the Leonardo board generates a compile error
   when using Serial.write(buf, len). The Teapot output uses this method.
   The solution requires a modification to the Arduino USBAPI.h file, which
   is fortunately simple, but annoying. This will be fixed in the next IDE
   release. For more info, see these links:

   http://arduino.cc/forum/index.php/topic,109987.0.html
   http://code.google.com/p/arduino/issues/detail?id=958
 *  ========================================================================
*/



// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
//#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal_lock)
//#define OUTPUT_READABLE_EULER
```

```
// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, us OUTPUT_READABLE_WORLDACCEL instead.
//#define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
//#define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
//#define OUTPUT_TEAPOT



#define INTERRUPT_PIN 2  // use pin 2 on Arduino Uno & most boards
#define LED_PIN 13 // (Arduino is 13, Teensy is 11, Teensy++ is 6)
bool blinkState = false;

// MPU control/status vars
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q;           // [w, x, y, z]         quaternion container
VectorInt16 aa;         // [x, y, z]            accel sensor measurements
VectorInt16 aaReal;     // [x, y, z]            gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]            world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]            gravity vector
```

```
float euler[3];         // [psi, theta, phi]    Euler angle container
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };



// ================================================================
// ===               INTERRUPT DETECTION ROUTINE                ===
// ================================================================

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone high
void dmpDataReady() {
  mpuInterrupt = true;
}



// ================================================================
// ===                      INITIAL SETUP                       ===
// ================================================================

void setup() {
  // join I2C bus (I2Cdev library doesn't do this automatically)
  #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    Wire.begin();
    Wire.setClock(400000); // 400kHz I2C clock. Comment this line if having compilation
difficulties
  #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    Fastwire::setup(400, true);
  #endif

  // initialize serial communication
  // (115200 chosen because it is required for Teapot Demo output, but it's
  // really up to you depending on your project)
  Serial.begin(115200);
  while (!Serial); // wait for Leonardo enumeration, others continue immediately

  // NOTE: 8MHz or slower host processors, like the Teensy @ 3.3v or Ardunio
  // Pro Mini running at 3.3v, cannot handle this baud rate reliably due to
  // the baud timing being too misaligned with processor ticks. You must use
  // 38400 or slower in these cases, or use some kind of external separate
```

```
    // crystal solution for the UART timer.

    // initialize device
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();
    pinMode(INTERRUPT_PIN, INPUT);

    // verify connection
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));

    // wait for ready
    Serial.println(F("\nSend any character to begin DMP programming and demo: "));
    //while (Serial.available() && Serial.read()); // empty buffer
    //while (!Serial.available());                 // wait for data
    while (Serial.available() && Serial.read()); // empty buffer again

    // load and configure the DMP
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // supply your own gyro offsets here, scaled for min sensitivity
    mpu.setXGyroOffset(-1173);
    mpu.setYGyroOffset(977);
    mpu.setZGyroOffset(1085);
    mpu.setXAccelOffset(22);
    mpu.setYAccelOffset(-28);
    mpu.setZAccelOffset(-26); // 1688 factory default for my test chip

    // make sure it worked (returns 0 if so)
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
        Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;
```

```
    // get expected DMP packet size for later comparison
    packetSize = mpu.dmpGetFIFOPacketSize();
  } else {
    // ERROR!
    // 1 = initial memory load failed
    // 2 = DMP configuration updates failed
    // (if it's going to break, usually the code will be 1)
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }

  // configure LED for output
  pinMode(LED_PIN, OUTPUT);
}



// ================================================================
// ===                    MAIN PROGRAM LOOP                     ===
// ================================================================

void loop() {
  // if programming failed, don't try to do anything
  if (!dmpReady) return;

  // wait for MPU interrupt or extra packet(s) available
  while (!mpuInterrupt && fifoCount < packetSize) {
    // other program behavior stuff here
    // .
    // .
    // .
    // if you are really paranoid you can frequently test in between other
    // stuff to see if mpuInterrupt is true, and if so, "break;" from the
    // while() loop to immediately process the MPU data
    // .
    // .
    // .
  }

  // reset interrupt flag and get INT_STATUS byte
  mpuInterrupt = false;
```

```
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024) {
  // reset so we can continue cleanly
  mpu.resetFIFO();
  Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)
} else if (mpuIntStatus & 0x02) {
  // wait for correct available data length, should be a VERY short wait
  while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

  // read a packet from FIFO
  mpu.getFIFOBytes(fifoBuffer, packetSize);

  // track FIFO count here in case there is > 1 packet available
  // (this lets us immediately read more without waiting for an interrupt)
  fifoCount -= packetSize;

  #ifdef OUTPUT_READABLE_QUATERNION
    // display quaternion values in easy matrix form: w x y z
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    Serial.print("quat\t");
    Serial.print(q.w);
    Serial.print("\t");
    Serial.print(q.x);
    Serial.print("\t");
    Serial.print(q.y);
    Serial.print("\t");
    Serial.println(q.z);
  #endif

  #ifdef OUTPUT_READABLE_EULER
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q);
    Serial.print("euler\t");
    Serial.print(euler[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(euler[1] * 180/M_PI);
```

```
    Serial.print("\t");
    Serial.println(euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
    // display Euler angles in degrees
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
    Serial.print("ypr\t");
    Serial.print(ypr[0] * 180/M_PI);
    Serial.print("\t");
    Serial.print(ypr[1] * 180/M_PI);
    Serial.print("\t");
    Serial.println(ypr[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_REALACCEL
    // display real acceleration, adjusted to remove gravity
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    Serial.print("areal\t");
    Serial.print(aaReal.x);
    Serial.print("\t");
    Serial.print(aaReal.y);
    Serial.print("\t");
    Serial.println(aaReal.z);
#endif

#ifdef OUTPUT_READABLE_WORLDACCEL
    // display initial world-frame acceleration, adjusted to remove gravity
    // and rotated based on known orientation from quaternion
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetAccel(&aa, fifoBuffer);
    mpu.dmpGetGravity(&gravity, &q);
    mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
    mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
    Serial.print("aworld\t");
    Serial.print(aaWorld.x);
    Serial.print("\t");
    Serial.print(aaWorld.y);
```

```
            Serial.print("\t");
            Serial.println(aaWorld.z);
        #endif

        #ifdef OUTPUT_TEAPOT
            // display quaternion values in InvenSense Teapot demo format:
            teapotPacket[2] = fifoBuffer[0];
            teapotPacket[3] = fifoBuffer[1];
            teapotPacket[4] = fifoBuffer[4];
            teapotPacket[5] = fifoBuffer[5];
            teapotPacket[6] = fifoBuffer[8];
            teapotPacket[7] = fifoBuffer[9];
            teapotPacket[8] = fifoBuffer[12];
            teapotPacket[9] = fifoBuffer[13];
            Serial.write(teapotPacket, 14);
            teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
        #endif

        // blink LED to indicate activity
        blinkState = !blinkState;
        digitalWrite(LED_PIN, blinkState);
    }
}
```

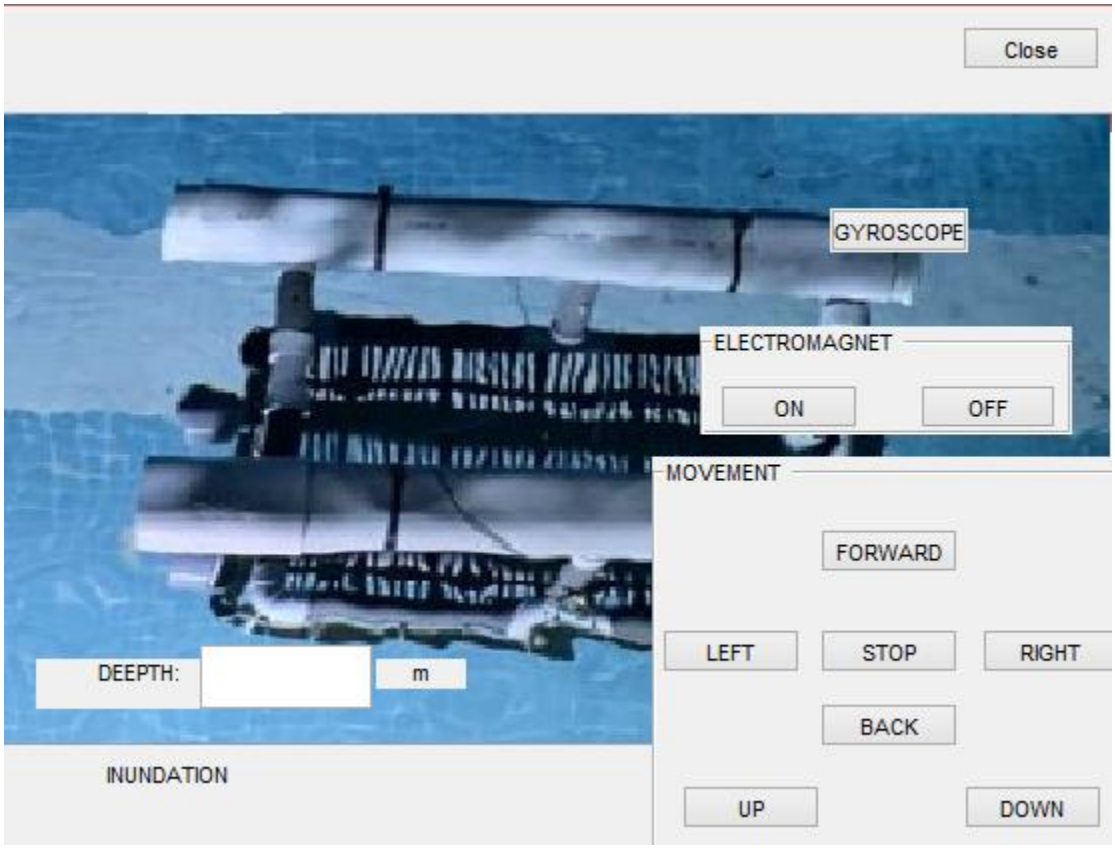
A destacar de este código que en las líneas 203 a 208 hemos de poner los valores de calibración que hemos obtenido anteriormente.

Los demás componentes del sumergible han sido programados directamente en *MATLAB*. Se ha hecho así para simplificar los programas y no tener que estar todo el rato con los 2 *software* trabajando.

2. Código de MATLAB

El programa consta de 2 partes, el archivo .m que contiene el *sketch,* y el archivo .fig que engloba la interfaz gráfica.

El primer paso fue realizar la interfaz gráfica con todos los botones, textos e imagen. La interfaz final corresponde a:



Luego, el código de la interfaz en archivo .m es:

```
function varargout = NAUTIL_control_panel(varargin)
% NAUTIL_control_panel MATLAB code for NAUTIL_control_panel.fig
%      NAUTIL_control_panel, by itself, creates a new NAUTIL_control_panel or
raises the existing
%      singleton*.
%
%      H = NAUTIL_control_panel returns the handle to a new
NAUTIL_control_panel or the handle to
%      the existing singleton*.
%
%      NAUTIL_control_panel('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in NAUTIL_control_panel.M with the given input
arguments.
%
%      NAUTIL_control_panel('Property','Value',...) creates a new
NAUTIL_control_panel or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before NAUTIL_control_panel_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to NAUTIL_control_panel_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help NAUTIL_control_panel

% Last Modified by GUIDE v2.5 24-Oct-2020 15:02:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @NAUTIL_control_panel_OpeningFcn, ...
                   'gui_OutputFcn',  @NAUTIL_control_panel_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end



function NAUTIL_control_panel_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;


guidata(hObject, handles);


function varargout = NAUTIL_control_panel_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

global a;
global s1;
global d1;
global aux;
global Vout;
instrfind;
a=arduino('COM3');
instrfind;
 aux=0;
for i= 0 : 100000
  %  k=readVoltage(a,'A3')
    pause(2);
    if readVoltage(a,'A3') <= 0.1
        set( handles.text7,'ForegroundColor','blue')
    else
        set( handles.text7,'ForegroundColor','red')
    end
    drawnow;
```

```
 aux =(readVoltage(a,'A0')*5.0/1023.0);
 Vout=aux/10;
 p = ( Vout / 0.0018*5 ) - (0.04 / 0.0018);
 h = ((p*1000)/(997*9.8))*100;
 set( handles.edit1,'String',num2str(p))

end
```

```
%ON
function pushbutton1_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;
            writeDigitalPin(a,'D12',1);
```

```
% OFF
function pushbutton2_Callback(hObject, eventdata, handles)
 global a;
global s1;
global d1;
            writeDigitalPin(a,'D12',0);
%FORWARD
function pushbutton3_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;


            for s1=0:0.1:0.5
            writePWMDutyCycle(a,'D3',s1);
            pause(1);
            end

            writeDigitalPin(a,'D5',0);
            for s1=0:0.1:0.5
            writePWMDutyCycle(a,'D6',s1);
            pause(1);
            end

            writeDigitalPin(a,'D9',0);
```

```
%LEFT
function pushbutton4_Callback(hObject, eventdata, handles)
 global a;
global s1;
global d1;

                writeDigitalPin(a,'D5',0);
                for s1=0:0.1:0.5
```

```
                writePWMDutyCycle(a,'D3',s1);
                pause(1);
                end

                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D9',s1);
                pause(1);
                end

                writeDigitalPin(a,'D6',0);


%RIGHT
function pushbutton5_Callback(hObject, eventdata, handles)
   global a;
global s1;
global d1;
                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D5',s1);
                pause(1);
                end

                writeDigitalPin(a,'D3',0);
                writeDigitalPin(a,'D9',0);
                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D6',s1);
                pause(1);
                end




% BACK
function pushbutton6_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;
                writeDigitalPin(a,'D3',0);
                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D5',s1);
                pause(1);
                end

                writeDigitalPin(a,'D6',0);
                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D9',s1);
                pause(1);
                end



%UP
function pushbutton7_Callback(hObject, eventdata, handles)
  global a;
global s1;
global d1;

                for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D6',s1);
                pause(1);
```

```
                end

                writeDigitalPin(a,'D9',0);


%DOWN
function pushbutton8_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;

             writeDigitalPin(a,'D6',0);

             for s1=0:0.1:0.5
                writePWMDutyCycle(a,'D9',s1);
                pause(1);
                end



% STOP
function pushbutton9_Callback(hObject, eventdata, handles)
global a;
global s1;
global d1;
                writeDigitalPin(a,'D3',0);
                writeDigitalPin(a,'D5',0);
                writeDigitalPin(a,'D6',0);
                writeDigitalPin(a,'D9',0);
                writeDigitalPin(a,'D10',0);
                writeDigitalPin(a,'D11',0);
                %pause(500000);
                s1=0;
                d1=0.5;
%clear all;
%clear global;
%a=arduino();




% CLOSE
function pushbutton11_Callback(hObject, eventdata, handles)
clear all;

closereq();


% --- Executes during object creation, after setting all properties.
function text5_CreateFcn(hObject, eventdata, handles)

% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


%giroscopio.
function pushbutton12_Callback(hObject, eventdata, handles)
global a;
while 1
x = fscanf(arduino, '%e');
y = fscanf(arduino, '%e');
plotCube(x, y);
drawnow
clf
end


% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


%Inundacion
function text7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

%background image
% --- Executes during object creation, after setting all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
imshow('imageROUV.jpg');
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3
```

A comentar del programa que la función general *varargout=handles.output* de las líneas 57 a 85 engloba los datos del sensor de inundación y sensor de presión. Los demás elementos se encuentran en *callback* propio.

El giroscopio, para poder visualizar la figura 3D, tiene un botón que abre otro archivo .m y .fig. Este archivo (CUBO_MPU6050.m) es el siguiente:

```
%% Initialization
close all;clear all;clc;
addpath('./Functions');

%% Create serial object for Arduino
baudrate = 115200;
%baudrate = 1200;
s = serial('COM3','BaudRate',baudrate); % change the COM Port number as
needed
s.ReadAsyncMode = 'manual';
set(s,'InputBufferSize',100);

%% Connect the serial port to Arduino
try
    fopen(s);
catch err
    fclose(instrfind);
    error('Make sure you select the correct COM Port where the Arduino is
connected.');
end

%% Prepare Figures

Fig = figure('Position',[9 9 913 1014],'ToolBar','none');
Ax(1) = axes('Position',[0.04 .81 0.92 .18]); grid; hold on;
xlabel('T');
ylabel('Ângulo (°)');

for k = 1:3
    H(k) = plot(0,0);
end

legend('Yaw','Roll','Pitch', 'Location','northeastoutside');

Ax(2) = axes('Position',[.15 0.03 .7 .7],'CameraPosition',[-9.1314 -11.9003
8.6603]);hold on;
axis([-1 1 -1 1 -1 1]);
grid;

%% Read and plot the data from Arduino
%Tmax = 60;Ts = 0.02; i = 1;ata = 0;t = 0;
%tic % Start timer
T(1)=0;
FLAG_CASTING = false;
CubH = [];Angles=zeros(1,3);
Flag_Initializing = true;
```

```
while(Flag_Initializing)
    while(strcmp(s.TransferStatus,'read'))
        pause(0.01);
    end
    readasync(s);
    sms = fscanf(s);
    if ~strcmp(sms(1:3),'ypr')
        fprintf(sms)
    else
        Flag_Initializing = false;
    end
end
while T(end) <= 2000
% while T(end) <= 100
    T(end+1)=T(end)+1;
    sms='a';
    idx = [];
    Angles = [0];
    while isempty(idx) || numel(Angles)~=3
        sms = fscanf(s);
        idx = find(sms=='r');
        if ~isempty(idx)
            idx = idx(end)+1;
            Angles = sscanf(sms(idx:end),'%f %f %f');
        end
    end
    Yaw = Angles(3);
    Pitch = Angles(2);
    Roll = Angles(1);

    k = 1;
    vY = get(H(k),'YData');vX = get(H(k),'XData');
    set(H(k),'YData',[vY,Angles(k)]);set(H(k),'XData',[vX,T(end)]);

    k = 2;
    vY = get(H(k),'YData');vX = get(H(k),'XData');
    set(H(k),'YData',[vY,Angles(k)]);set(H(k),'XData',[vX,T(end)]);

    k = 3;
    vY = get(H(k),'YData');vX = get(H(k),'XData');
    set(H(k),'YData',[vY,Angles(k)]);set(H(k),'XData',[vX,T(end)]);

    CubH = Plot_Cubo(deg2rad(-Yaw),deg2rad(Pitch),deg2rad(Roll),Ax(2),CubH);
    drawnow;

end
fclose(s);
```

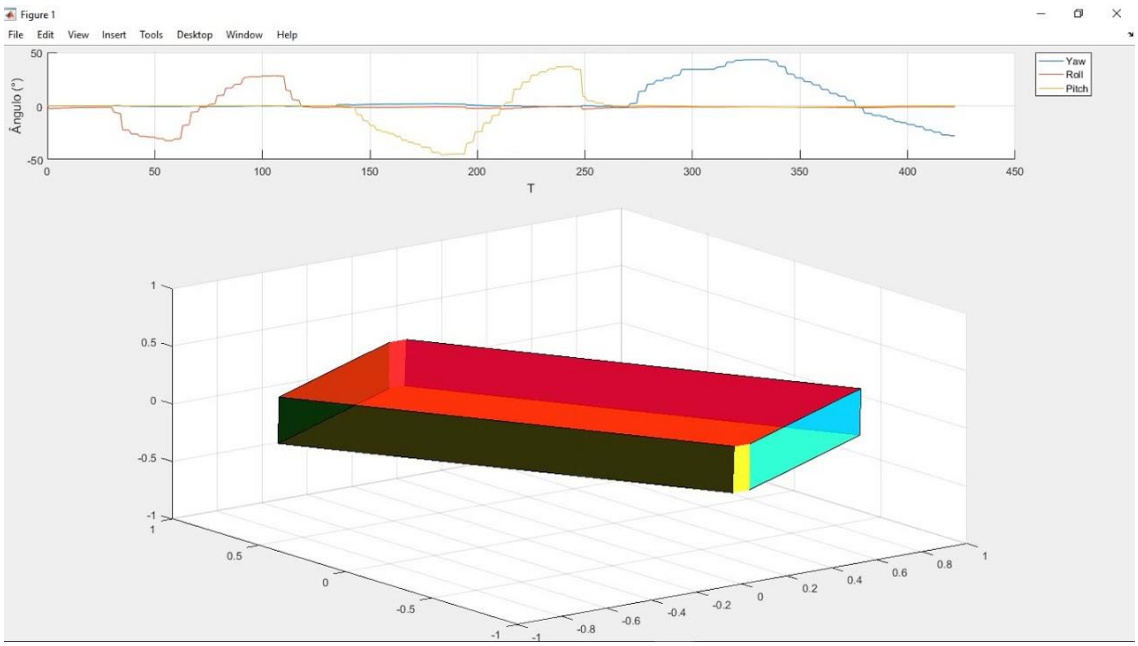
Como puede apreciarse en la línea 88, se hace referencia a un archivo llamado *Plot_Cubo*, el cual delimita la figura 3D que irá dentro de la gráfica y representará a nuestro sumergible.

```
function CUBO = Plot_Cubo(yaw,pitch,roll,Ax,CUBO)
if ~exist('CUBO','var')
    CUBO = [];
```

```
end
C=[0 0 0];
LW = 0.1;
%Height
H = 0.4;
%Width
W = 1.8;
%Depth
D = 1;
R{3} = [1 0 0;0 cos(yaw) -sin(yaw);0 sin(yaw) cos(yaw)];
R{1} = [cos(pitch) 0 sin(pitch);0 1 0;-sin(pitch) 0 cos(pitch)];
R{2} = [cos(roll) -sin(roll) 0;sin(roll) cos(roll) 0;0 0 1];
    if ~exist('Ax');
        F = figure;
        Ax = axes('CameraPosition',[-9.1314 -11.9003 8.6603]);
        hold on;
    end

    Vertex=[-D -W -H;
            +D -W -H;
            +D +W -H;
            -D +W -H;
            -D -W +H; % Muda
            +D -W +H;
            +D +W +H;
            -D +W +H]/2;
    for rot = 1:3
        for v = 1:8
            Vertex(v,:) = Vertex(v,:)*R{rot};
        end
    end
    % Base
    idx1 = [1 2 3 4 1];
    % Topo
    idx2 = [5 6 7 8 5];
    % Lados
    idx3 = [1 5 6 2];
    idx4 = [2 6 7 3];
    idx5 = [3 7 8 4];
    idx6 = [4 8 5 1];

    if isempty(CUBO)
        CUBO(1) = fill3(Vertex(idx1,1),Vertex(idx1,2),Vertex(idx1,3),[1 1
0]);
        CUBO(2) = fill3(Vertex(idx2,1),Vertex(idx2,2),Vertex(idx2,3),[1 0
0]);
        CUBO(3) = fill3(Vertex(idx3,1),Vertex(idx3,2),Vertex(idx3,3),[0 1
1]);
        CUBO(4) = fill3(Vertex(idx4,1),Vertex(idx4,2),Vertex(idx4,3),[0 0
1]);
        CUBO(5) = fill3(Vertex(idx5,1),Vertex(idx5,2),Vertex(idx5,3),[0 1
0]);
        CUBO(6) = fill3(Vertex(idx6,1),Vertex(idx6,2),Vertex(idx6,3),C);

        alpha(0.8);
    else

set(CUBO(1),'XData',Vertex(idx1,1),'YData',Vertex(idx1,2),'ZData',Vertex(idx1
,3));
```

```
set(CUBO(2),'XData',Vertex(idx2,1),'YData',Vertex(idx2,2),'ZData',Vertex(idx2
,3));

set(CUBO(3),'XData',Vertex(idx3,1),'YData',Vertex(idx3,2),'ZData',Vertex(idx3
,3));

set(CUBO(4),'XData',Vertex(idx4,1),'YData',Vertex(idx4,2),'ZData',Vertex(idx4
,3));

set(CUBO(5),'XData',Vertex(idx5,1),'YData',Vertex(idx5,2),'ZData',Vertex(idx5
,3));

set(CUBO(6),'XData',Vertex(idx6,1),'YData',Vertex(idx6,2),'ZData',Vertex(idx6
,3));

    end


end
```

El archivo .fig de CUBO_MPU6050.m , que contiene 2 gráficas, una 2 D y otra 3D es el siguiente:



ANEXO 2. Instrucciones de usuario

Para un correcto y seguro funcionamiento síganse las normas:

1. Seguridad

Observar que ningún cable del umbilical se haya desgarrado y que pueda provocar una electrocución al tocarlo.

No manipular el aparato cuando estén los motores funcionando.

Vigilar con usar la batería en sitios húmedos.

Alejarlo del alcance de los niños.

En caso de cambiar el fusible de la caja de control, utilizar los porta fusibles roscados y desconectar la batería.

2. Operativa

A seguirse los siguientes pasos:

1. Realizar un chequeo preliminar.
2. Poner el sumergible en el agua. Gracias a su leve flotabilidad positiva se mantendrá en la superficie.
3. Comprobar que ningún cable del umbilical obstaculice las hélices de los motores.
4. Conectar los bornes de la alimentación a la batería.
5. Encender la cámara.

6. Desplegar el umbilical enrollado.

7. Escoger método de operación.
   Si se selecciona el mando, puede procederse al arranque de los motores mediante los interruptores dobles.
   Si por lo contrario se encoge utilizar la interfaz gráfica para tener un control más suave del movimiento, recibir datos de profundidad, alertas, electroimán y estabilidad, encender el *software MATLAB,* conectar el cable USB, cargar el programa de la interfaz gráfica y operar el aparato.

8. Una vez terminada la operación, hacer ascender y acercar el aparato.

9. Desconectar los bornes de la batería.

10. Recoger el umbilical y retirar el *ROUV* del agua.

11. Una vez seco, puede guardarse hasta su posterior uso.

3. Conservación

Mantener durante el reposo el aparato en un lugar seco, a temperatura ambiente y que no le dé directamente el sol.

No abrir las cajas estanca ni de control. En caso de aparecer algún fallo no representado en la hoja de análisis de fallo, consultar al fabricante.

Tras su inmersión, comprobar el estado del sumergible antes de guardarlo. En caso de haber funcionado en agua salada, pasarle un trapo con agua dulce y dejarlo secar.

Cargar periódicamente las baterías.

ANEXO 3. Ficha de análisis de fallos

A continuación se presenta en formato ficha los posibles fallos que pueden tener el prototipo y sus soluciones.

Cabe destacar que existen más fallos que pueden pasar en el aparato. Esta ficha simplemente recoge los principales y más frecuentes.

Componente	*Payload-Box*	Comunicación *ARDUINO-MATLAB*	Cámara no funciona	Algún sensor no funciona
Explicación del fallo	Entrada de agua en la caja donde se ubican todos los componentes eléctricos del *ROUV*	Bien sea por no disponer de ordenador o el programa *MATLAB* instalado.	Podría faltar cargar su batería independiente.	Podría ser debido a un cortocircuito del cableado de control. Los sensores son sensibles y pueden quemarse fácilmente debido a ello.
Gravedad	Mayor	Menor	Menor	Media
Acciones propuestas	En caso de que sea una inundación pequeña, el sensor de inundación avisará al operario y este tendrá tiempo de hacer ascender al *ROUV* evitando posibles daños mayores. En caso de que la entrada de agua sea importante y falle la electrónica, habrá que subir manualmente al submarino mediante el cabo.	Proceder a realizar las maniobras del sumergible de forma mecánica (mediante la caja de control y los interruptores) Así mismo. Se puede ver la imagen emitida por la cámara a tiempo real.	Cargar su batería con el cargador proporcionado. En caso de que siga sin funcionar, mandar al fabricante.	En caso de tener los conocimientos suficientes. Abrir la *payload box* y comprobar tanto el cableado como el propio sensor para ver si se detecta alguna anomalía y proceder a la reparación o cambio. En caso de no tener los conocimientos necesarios, mandar el equipo al fabricante.
Elementos de seguridad para evitar el fallo	Sensor de inundación, cabo atado al control y al vehículo, mirilla de inspección visual para corroborar que no haya entrada de agua en la caja eléctrica.	Fusible de 10 A. En caso de cortocircuito cambiar el fusible.		Antes de cerrar todo el equipo eléctrico, se realizan varias comprobaciones tanto de funcionamiento como de la soldadura de los pines y cables.